# DEVELOPMENT OF ROBOSIM FOR ACADEMIC/INDUSTRIAL USE

## - FINAL REPORT -


**Prepared by**

**George E. Cook, PI**
**Csaba Biegl**
**James F. Springfield, Jr.**

**Vanderbilt University**
**School of Engineering**
**Box 1826, Station B**
**Nashville, TN 37235**

**Telephone: (615) 322-2764**
**FAX: (615) 343-8006**
**E-mail: cookge@vuse.vanderbilt.edu**





**Grant No. NAG8-116**

# DEVELOPMENT OF ROBOSIM FOR ACADEMIC/INDUSTRIAL USE



## FINAL REPORT

NASA Grant No. NAG8-116

Vanderbilt University

## SUMMARY

The objectives of this research were to provide enhancements to a graphical simulation system, ROBOSIM, consisting of additional features considered essential for many applications, to port the package into a form compatible with lower cost and more useful engineering workstations, and to develop learning aids and documentation for users of the software package. In order to remain fully compatible with its original FORTRAN-coded, mainframe computer implementation, the facilities of ROBOSIM were expanded without changing its structure. This was accomplished by porting the FORTRAN code to C, keeping ROBOSIM as the kernel, and adding features to the front and back ends.

Features added to the simulator include a modeling environment and an interactive simulation environment plus a number of advanced capabilities. The advanced features of the simulator include: *composite objects* supporting the linking of separate objects into a composite object which can be operated on as a unit while preserving the separate accessibility of its components; *configuration management* with the provision of an automatic configuration selection mechanism based on a set of heuristic rules; *collision detection* which provides a way to check for collisions during a simulation run; *collision avoidance* where the simulator provides a heuristic path planning algorithm which is capable of recovering from collision situations to provide collision-free path plans; and *interface to control physical robots* with the capability of generating command sequences for real robot controllers.

Implementations of the simulator have been developed for the Hewlett Packard 9000/300 and 9000/800 graphics workstation families, Silicon Graphics workstations, Intergraph workstations, and IBM compatible PCs using Intel 80386 and higher CPUs with EGA or VGA displays. The display mode may be wire-frame or shaded solid modeling for the workstation implementations. Currently, the PC version displays in wire-frame mode only. While currently limited to wire-frame displays, the PC version is quite fast, offering displayed manipulator motions at speeds normally well in excess of the hardware being simulated.

Included in this report is a Manual, Tutorial, and Installation Instructions for installing and using ROBOSIM and disks containing the program and all examples.

# TABLE OF CONTENTS

## LISTINGS (MODELING AND SIMULATION CODE)

# LIST OF FIGURES

# NOMENCLATURE

| | |
|---|---|
| CAD | Computer Aided Design |
| CPU | Central Processing Unit |
| EGA | Enhanced Graphics Adapter |
| GUI | Graphical User Interface |
| OMV | Orbital Maneuvering Vehicle |
| PC | Personal Computer |
| ROBOSIM | Robot Simulation System |
| SRB | Solid Rocket Booster |
| SSME | Space Shuttle Main Engine |
| VGA | Video Graphics Adapter |

# EXECUTIVE SUMMARY

## Objectives:

The objectives of this research were to provide enhancements to a graphical simulation system, ROBOSIM, consisting of additional features considered essential for many applications, to port the package into a form compatible with lower cost and more useful engineering workstations, and to develop learning aids and documentation for users of the software package. In order to remain fully compatible with its original FORTRAN-coded, mainframe computer implementation, the facilities of ROBOSIM were expanded without changing its structure. This was accomplished by porting the FORTRAN code to C, keeping ROBOSIM as the kernel, and adding features to the front and back ends.

## Research Conducted:

Features added to the simulator include a modeling environment and an interactive simulation environment plus a number of advanced capabilities. The advanced features of the simulator include: *composite objects* supporting the linking of separate objects into a composite object which can be operated on as a unit while preserving the separate accessibility of its components; *configuration management* with the provision of an automatic configuration selection mechanism based on a set of heuristic rules; *collision detection* which provides a way to check for collisions during a simulation run; *collision avoidance* where the simulator provides a heuristic path planning algorithm which is capable of recovering from collision situations to provide collision-free path plans; and *interface to control physical robots* with the capability of generating command sequences for real robot controllers.

## Implementations:

Implementations of the simulator have been developed for the Hewlett Packard 9000/300 and 9000/800 graphics workstation families, Silicon Graphics workstations, Intergraph workstations, and IBM compatible PCs using Intel 80386 and higher CPUs with EGA or VGA displays. The display mode may be wire-frame or shaded solid modeling for the workstation implementations. Currently, the PC version displays in wire-frame mode only. While currently limited to wire-frame displays, the PC version is quite fast, offering displayed manipulator motions at speeds normally well in excess of the hardware being simulated.

## User Aids:

Included in this report is a Manual, Tutorial, and Installation Instructions for installing and using ROBOSIM and disks containing the program and all examples.

## 1.0 INTRODUCTION

ROBOSIM is a graphical simulation system used for three-dimensional geometrical modeling of robot manipulators and various objects in their workspace, and for the simulation of action sequences performed by the manipulators. ROBOSIM was originally conceived and developed between 1985-88 by NASA-MSFC and Vanderbilt University as a means of rapid prototyping of robotics systems [1]. This early version of the simulation system was written in FORTRAN and was developed for DEC VAX mainframe computers with TEKTRONIX 4014 terminals and also Evans & Sutherland graphics terminals. Robots and other objects in the simulated environment were specified through a *program* that the user wrote. After a robot had been designed, the user could solve the inverse kinematics problem for a set of points and store the results in a file. This file could then be read back in and used to display the robot at the stored points. The TEKTRONIX terminal provided a low-cost, low-speed way to view the robot while the Evans & Sutherland terminal allowed high-speed display. However, both could only display wire-frame models.

The original version of ROBOSIM was basically a robot modeling tool. It did not contain a simulator for operating, from high-level commands, the robots modeled. It also did not contain collision detection capabilities, and it did not possess any advanced features of artificial intelligence-based collision avoidance or path planning capabilities. Despite these shortcomings, ROBOSIM was used at the MSFC in a number of space related activities including: development of off-line welding programs for robots used in fabricating the Space Shuttle Main Engine (SSME), integration of a vision sensor into a robotics workcell used for Solid Rocket Booster (SRB) refurbishment, and planning of a robot mechanism to be mounted on the Orbital Maneuvering Vehicle (OMV). At Vanderbilt University, ROBOSIM was used in robot manipulator classes and as a research tool for robot task planning.

The advent of high-speed graphics workstations provided the impetus to port ROBOSIM to these engineering platforms. Also, it was recognized that a number of functional enhancements would be necessary if ROBOSIM were to find widespread usefulness in the industrial and academic environments. To meet these goals, the subject grant *Development of ROBOSIM for Academic/Industrial Use* was awarded to Vanderbilt University. The objectives of the grant were to enhance ROBOSIM with additional features considered essential for many applications, to develop learning aids and documentation for users of the software package, and to port the package into a form compatible with lower cost and more useful engineering workstations.

This Final Report describes the results of this work. In order to remain fully compatible with the VAX implementation, it was desired to expand the facilities of ROBOSIM without changing its structure. This has been accomplished by porting the FORTRAN code to C, keeping ROBOSIM as the kernel, and adding features to the front and back ends [2,3]. These features include a modeling environment and a simulation environment and advanced features including composite objects, configuration

management, collision detection (see Appendix A), collision avoidance, and interface to physical robots. Implementations of the simulator have been developed for the Hewlett Packard 9000/300 and 9000/800 graphics workstation families, Silicon Graphics workstations, Intergraph workstations, and IBM compatible PCs using Intel 80386 and higher CPUs with EGA or VGA displays. The display mode may be wire-frame or shaded solid modeling for the workstation implementations. Currently, the PC version displays in wire-frame mode only. While currently limited to wire-frame displays, the PC version is quite fast, offering displayed manipulator motions at speeds well in excess of the hardware being simulated.

This report describes the new intelligent simulation system that has been developed in terms of ROBOSIM's modeling environment and interactive simulation environment. Commands used in the modeling and simulation environments are described in the Manual included in Appendix B. Appendix C contains a Tutorial for getting started in using the simulation system, and Appendix D contains Installation Instructions. Appendix E contains examples of code and graphic display for various simulations. A disk containing the executable code, manual, tutorial, and installation instructions is attached to the rear inside cover of this manual.

## 2.0 BACKGROUND

Geometrical modeling of robot manipulators is an expanding area of research because it can aid in the design and usage of robots in a number of ways [4]:

- *Design and testing of manipulators:* The purpose of the modeling is to study different approaches to satisfy the design specifications of the manipulator.

- *Robot action planning:* The modeling environment is used to build a representation of the robots, positioners, and other mechanisms with moving joints, and the objects in the workspace for creating and validating action plans by simulating the effect of these actions in the model space.

- *On-line control of robot manipulators:* The simulated action plans generated in the model space are transmitted (after validation) to the attached robot manipulators for execution.

- *Training and education:* Robotics simulation packages provide an inexpensive and safe way to teach the theory and operation of robot manipulators.

- *Telerobotic user interface:* In applications where the operator of the robot has to be at a large distance from the workcell (radiation, space, etc.) realistic graphical simulation can be used for better interaction with the manipulator.

To satisfy all or some of the above goals, a robotics modeling and simulation package has to provide the following minimal features:

- A way to build models of solid objects. This requires the facilities to create a set of solid primitives (like boxes, cylinders, cones, etc.) and some services to combine these into more complex shapes.

- A way to add the necessary kinematic (an possibly dynamic) information to the above models of solid objects to turn them into robot manipulator links.

- A way to assemble models of robot manipulator links and other solid objects into complete models of robotic work cells.

- A way to manipulate the models. This includes methods for accessing and animating objects, routines for forward and inverse kinematics calculations, and graphics display.

Advanced robot simulation environments can also support one or more of the following:

- Path planning services for moving the manipulator along various trajectories. (straight-line, etc.)

- Collision detection and collision avoidance services.

- Simulation of manipulator dynamics to obtain the forces and torques on the links of the manipulator arm and to enforce their limits.

The following sections of this report describe how some of the features described above have been implemented in the ROBOSIM package.

# 3.0 CREATION OF ROBOT MODELS, POSITIONERS AND OBJECTS: THE ROBOSIM MODELING ENVIRONMENT

Models of robot manipulators, positioners, and other mechanisms with moving joints, and objects that make up the complete simulated environment are constructed in ROBOSIM's *modeling environment.* ROBOSIM also provides a simulation environment where every command entered by the user is executed and the results are displayed on a graphics screen. From this interactive environment users can change the simulation scenario and operate the robot manipulator models previously constructed in the modeling environment. This section describes the modeling environment. The simulation environment is described in section 4.0.

Similarly to other solid modeling software tools, ROBOSIM models the three dimensional geometric objects using lists of their bounding polygons. The ROBOSIM modeling language is used to specify complex geometric shapes which are used as manipulator links or as passive cylinders, cones, extruded polygons, etc. Translational, rotational and scaling transformations are used to combine these objects to form the desired shape.

The ROBOSIM modeling language mimics a "geometric microprocessor" [1] which has a few graphics registers ("a" through "d") containing the three dimensional polygon lists of the complex object being built. There is a designated accumulator register ("a"). Whenever, a new elementary shape (box, etc.) is created, it is concatenated to the contents of the accumulator register. Similarly to the basic shape generation primitives, the geometrical transformations also operate on the contents of the accumulator register. This, together with the ability to move or concatenate geometric data to or from the other registers enables the definition of arbitrary complex shapes in terms of a few basic operations. Link coordinate frames can also be added to the contents of the accumulator register thus making it possible to specify the geometric transformations associated with the links of a manipulator. The ROBOSIM modeling language is described below.

## 3.1 Modeling Commands

### 3.1.1 Solid primitive commands

All solid objects are created with their center of mass at the origin of the world coordinate system and their principal axis parallel with the z-axis. The commands are:

**box x=X y=Y z=Z**
**cylinder r=R h=H**
**cone r=R h=H**
**sphere r=R**

In each case, the solid primitive is generated in the "a" register with appropriate mass properties. In each of the commands, the order of the arguments is up to the user.

### 3.1.2 Graphic register control commands

Contents of the graphic registers are manipulated with the following commands:

| | |
|---|---|
| **clear** | Clears the contents of the "a" graphics register. |
| **store b or c or d** | The contents of the "a" graphics register are stored in one of the temporary graphics registers (b,c,d). The contents of the "a" register are unchanged. |
| **load b or c or d** | The contents of the b or c or d register are copied into the "a" graphics register destroying its previous contents and leaving the b or c or d register unchanged. |
| **exchange b or c or d** | The contents of the b or c or d register are exchanged with the contents of the "a" graphics register. |
| **add b or c or d** | The contents of the b or c or d register are appended to the contents of the "a" graphics register leaving the b or c or d register unchanged. |

### 3.1.3 Surface generation commands

Surfaces are generated with the following commands:

| | |
|---|---|
| **vector x=X y=Y z=Z** | Generates a vector from the origin to (X,Y,Z) in the "a" graphics register. |
| **move x=X y=Y z=Z** | Generates a non-visible vector from the previous vector tip to (X,Y,Z) in the "a" graphics register. |
| **draw x=X y=Y z=Z** | Generates a visible vector from the previous vector tip to (X,Y,Z) in the "a" graphics register. |
| **rev-surface** | Generates a surface by revolving the contents of the "a" register about the z-axis. |
| **extrude-surface z=Z** | Generates a surface by sweeping the graphical contents of the "a" register along the z-axis. |

### 3.1.4 Model manipulation commands

The primitive elements of the model are manipulated with the following commands:

**translate x=X y=Y z=Z** Translates the graphical contents of the "a" register the specified relative distance.

**rotate x=X or y=Y or z=Z** Rotates the graphical contents of the "a" register by the specified angle in degrees about the x, y, or z axes of the world coordinate frame.

**scale x=X y=Y z=Z** Multiples the graphical contents of the "a" register by the specified amounts along the x, y, and z axes.

### 3.1.5 File control commands

**execute-file fname** The ROBOSIM commands contained in the external file, fname, are placed in place of this instruction. All commands are allowed except "execute-file" itself.

**store-file fname** Stores the graphical and mass properties contained in the "a" register into the file (fname) specified. The contents of the "a" register are unchanged.

### 3.1.6 Link joint specifications

**(r, p, or f)-joint-(i or i+1)** A revolute, prismatic, or fixed joint is specified by the prefix r, p, or f, respectively. The suffix "i" indicates the joint is closer to the robot base than the joint specified with suffix "i+1".

### 3.1.7 Special commands

**end** Terminates execution of the ROBOSIM process.

**set-nface n=N** Sets the number of facets generated to approximate a circular surface. This specification affects the solid primitive routines and rev-surface command.

**set-density d=D** Sets the default mass-density for solids defined by the primitives, extrude and rev-surface command.

## 3.2 Modeling Example

Listing 1 shows an example ROBOSIM source listing which models a link of the PUMA 560 manipulator arm. Other links of the arm are modeled similarly.

```
;*********************
; Base Link - l0
;*********************
clear
cylinder r=88.90 h=533.40
translate x=0.00 y=0.00 z=328.93
store b
clear
cylinder r=177.80 h=25.40
translate x=0.00 y=0.00 z=49.53
add b
store b
clear
cylinder r=203.20 h=36.83
translate x=0.00 y=0.00 z=18.41
f-joint-i
add b
store b
clear
cylinder r=38.10 h=152.40
translate x=0.00 y=0.00 z=127.00
box x=127.00 y=101.60 z=101.60
translate x=0.00 y=152.40 z=50.80
add b
store b
clear
r-joint-i+1
translate x=0.00 y=0.00 z=671.83
add b
store-link puma560.l0
```

Listing 1. ROBOSIM Model of PUMA 560 Link

Once the ROBOSIM language interpreter finishes the processing of the code describing a solid shape, the resulting polygon list in the accumulator register is converted into a named object in the robot simulator package's workspace. Thus the modeling of robot arm and geometric scenarios is a two step process as follows:

- modeling the geometric shapes, robot links, etc., which are used to build the scenario.

- creating one or more named, distinguishable object instances of these shapes in the simulator's workspace. These object instances can be individually manipulated upon, moved around, etc., from the interactive environment of the simulator.

Figure 1 shows the structure of the ROBOSIM modeling environment described above, while Listing 2 shows an example command sequence which can be used to set up the scenario shown in Figure 2.

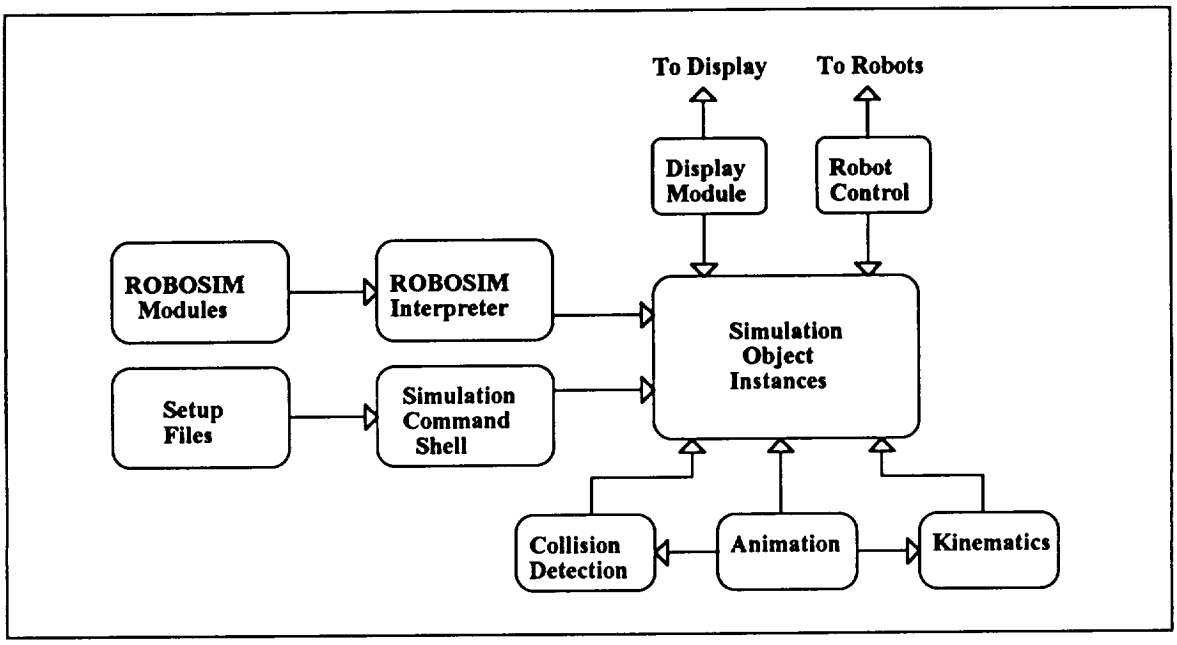


Figure 1. ROBOSIM modeling environment.

```
;*********************
;
;        SETUP.CMD
;*********************
;
define-color red 1 0 0;                    COLOR DEFINITIONS
define-color green 0 1 0
define-color blue 0 0 1
look-at 0 0 700;                           VIEW COMMANDS
look-from 0 5000 1000
make-agent r puma560 red;                  ROBOT - R
make-object floor box 2000 2000 60 green;  FLOOR
translate-object floor z=-31
make-object xx box 200 10 100 blue;        OBJECT - XX
define-grasping-point xx -180 0 0 -90 0 -90
translate-object xx x=900 y=300 z=800
make-object top cylinder 200 10 blue;      TOP - CYLINDER
translate-object top z=1650
end
```

Listing 2. Configuration Commands for Figure 2.

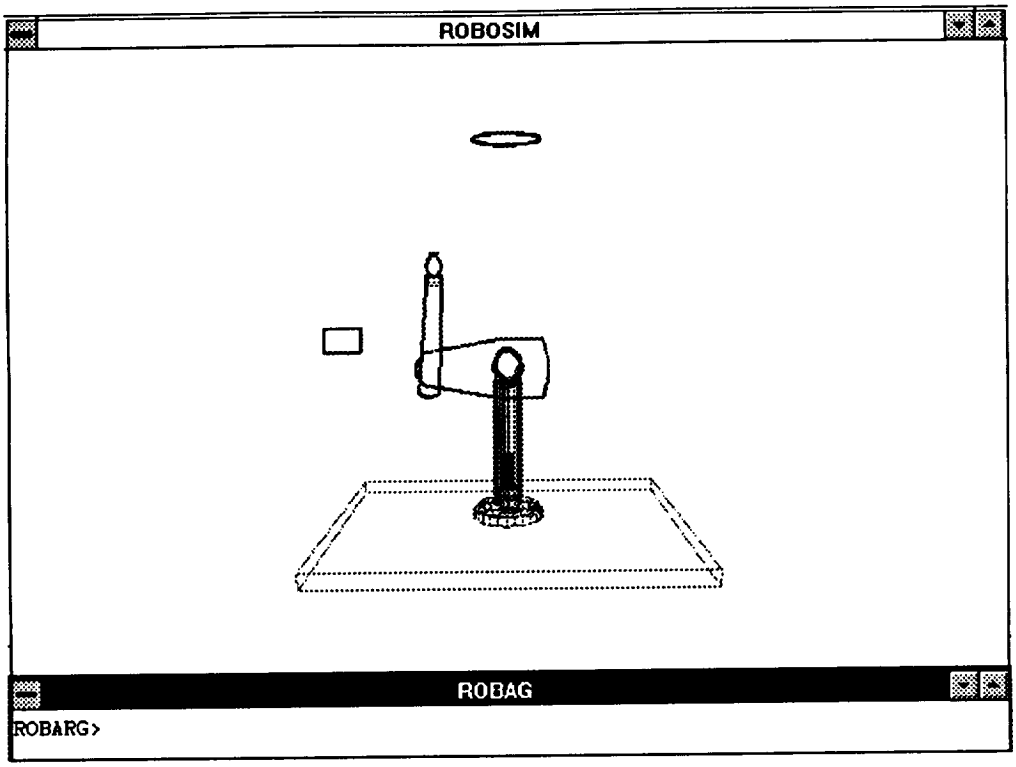


Figure 2. An example ROBOSIM screen.

# 4.0 OPERATING THE MODELS: THE ROBOSIM SIMULATION ENVIRONMENT

The ROBOSIM package provides an interactive simulation environment where every command entered by the user is immediately executed and the results are displayed on a graphics screen. From this interactive environment users can change the simulation scenario and operate the robot manipulator models in the system. The commands available can be grouped as follows:

- *Environment configuration:* Besides the modeling services discussed above additional commands are available for the setting of global parameters like camera position, display mode, light source, etc. The graphics display module of the simulator supports different display options like wireframe, hidden line, solid filled and shaded graphics depending on the capabilities of the hardware platform.

- *Manipulator control:* There are commands available for moving the models of manipulator arms in various modes: joint interpolated, straight-line, rotation about an arbitrary axis, etc. Manipulator coordinates can be specified both in joint and world coordinates. The simulator has a built-in interactive inverse kinematics algorithm, but the user can also specify an explicit inverse kinematics method for his or her manipulator if such a method is available. Additionally, the objects in the workspace can be grasped, moved and released by the robots. If the scenario contains several manipulators these can be operated in parallel.

- *Status reporting:* Reports about different aspects of the simulator's operation (arm position, collision situations, etc.) can be obtained by using one of the appropriate commands from this group.

The command language of the ROBOSIM simulation environment has been designed with two goals in mind: (1) to provide an interactive user interface, and (2) to be usable as the interface to a higher-level task planner program. In the second application the task planner and the robot simulator are typically interfaced using some kind of pipe mechanism and the task planner outputs similar command sequences as entered by users in interactive applications. For this reason the command language has intentionally been kept simple. A description of each of the ROBOSIM simulation environment commands is given below.

## 4.1 Simulation Commands

### 4.1.1 Command format

The simulator uses a character stream command protocol. Commands can be entered from the system console, loaded from a file, or sent by another program using the pipe mechanism provided by the operating system. The general format is:

**[<label>:] <command> <arg1> ... <argN> [; comment]**

Multiple agent (robot) movement commands per command line are possible. The individual commands are separated by commas. Each of the commands may be directed to a different agent in the system, which will execute the commands in parallel. The execution of the next command line begins when the last agent finishes its operation. In contrast, if commands to two different agents are placed into consecutive command lines, the execution of these commands will be sequential.

### 4.1.2 Error reporting

While processing a command stream, the simulator generates an error log stream. For each command line which could not be completed successfully, there will be an entry in this stream. The format of this entry is the following:

***** Error in line [label:NNN|NNN] --- <code> <message>**

If there were a label preceding the command containing the error, then the error message will contain the name of the last seen label and the number of lines read since the last label was encountered. If the command stream did not contain labels, then the error message will contain the number of the command line counted from the beginning of the stream.

### 4.1.3 Object creation commands

The following object creation command creates an object in the workspace of the simulator:

**make-object <object name> <object type> <parameters>**

Various object types have been defined, they and their parameters are described below.

**box <xsize> <ysize> <zsize> [<color>]**
**cylinder <radius> <height> [<color>]**
**cone <radius> <height> [<color>]**
**truncated-cone <radius1> <radius2> <height> [<color>]**
**sphere <radius> [<color>]**

All solid objects are created with their center of mass at the origin of the coordinate system and their principal axis parallel with the z-axis. The optional color parameter is the

name of a user defined color (see later). All other parameters are numbers. The simulator does not make any assumptions about the physical units used, it is the responsibility of the user to specify the sizes of each object in a coherent way.

### 4.1.4 Object transformation commands

The object transformation commands available in the simulator are:

**translate-object <object> x=<xtran> y=<ytran> z=<ztran>**
**rotate-object <object> x=<xrot> y=<yrot> z=<zrot>**

These commands have a slightly different argument structure which serves the purpose of using defaults. If any coordinate direction is missing from the arguments, it is assumed to be 0. The order of the arguments is up to the user. Transformations are performed in the order of the arguments in the argument list.

### 4.1.5 Composite objects

The composite object commands are:

**make-composite-object <name> <object1> <object2> ...**
**link-objects <name> <object1> <object2> ...**

These commands create a new composite object by joining the objects in the argument list permanently. They do not perform any transformations on the argument objects, but simply use their current positions. The first command will create a new object and leave the components in the workspace, while the second one will remove all components from the workspace after creating the composite object. Composite objects can be used (transformed, operated on by agents, etc.) in a manner identical to the elementary objects.

### 4.1.6 Objects created with ROBOSIM modeling language

The following command can be used to create a composite object using its ROBOSIM source code written in the modeling language described in section 3.0:

**make-robosim-object <name> <filename> <objectname> [<color>]**

The specified modeling language file is scanned until a 'store-file' command is encountered with the specified object name as its argument. An optional color parameter is also accepted.

### 4.1.7 Agents

The following command creates an agent:

**make-agent <name> <agent-type> [<color>]**

Agents are basically robots whose models have been pre-compiled using the ROBOSIM modeling language described in section 3.0.

### 4.1.8 Agent positioning

The following commands may be used to move (position) the agent in the simulation environment.

**drive <agent> <joint angles> | <joint angle vector name>**
**move-straight <agent> <coordinates>**
**move-inter <agent> <coordinates>**
**find-path <agent> <coordinates>**
**move-straight-to <agent> <object> <coordinates>**
**move-inter-to <agent> <object> <coordinates>**
**find-path-to <agent> <object> <coordinates>**

The coordinate specifications are agent specific, for the PUMA they must contain six values of either joint angles (only for the drive command) or rectangular coordinates (x, y, z, roll, pitch, yaw) either workspace absolute or object relative. The movement can be straight-line or joint-interpolated, or the agent can be instructed to *find a path* to the desired location based on its knowledge of the workspace configuration and built-in collision-free path-planning heuristics. The drive command also accepts a previously recorded joint angle vector (see later).

Two additional agent positioning commands that are useful when small incremental motions of the robot arm are needed are:

**translate-agent <agent> x=<xtran> y=<ytran> z=<ztran>**
**rotate-agent <agent> x=<xrot> y=<yrot> z=<zrot>**

These commands always perform straight-line motion. The coordinate specification uses the same scheme as the object transformation commands, that is the coordinates are named, and any unspecified coordinate direction is assumed to be 0.

The robot movement simulation step size for the movement commands is set with the command:

**minimal-step <value>**

The inverse kinematics solution used for the agent is set by the command:

**set-solution <agent> <value>**

The inverse kinematics method for the PUMA 560 arm provides 8 different solutions for (almost) any location. Some of these solutions are typically invalid due to joint angle constraints. The accepted range for 'value' is 0...7. The command also accepts the special value of -1 which instructs the agent to select the most suitable solution automatically. This is the default operation of the agent. Setting a fixed configuration index will more likely result in joint limit violation error messages, since the agent has no chance for switching solutions. For small movements the automatic selection is based upon choosing a valid configuration which is closest to the current joint variable values. This strategy works best when the agent is performing various tasks in a relatively small part of the workspace. However for a major position shift this may not be the best approach. In such cases the agent will select the new configuration which offers the most room (i.e., all joint angles are as far from their respective limits as possible) for moving around in the vicinity of the new location. To determine which strategy to use the agent compares the joint angles of the old and new positions. If the difference is larger than a preset threshold, the second method is used, otherwise the first. This threshold value can be set with the following command (default value is 45 degrees):

**set-large-move-limit <limit>**

In most cases this strategy works fine. However, it is possible that in some cases explicit control of the robot arm configuration is necessary. (It is most likely to occur if relatively large straight-line motion segments are needed. In straight-line motion mode the simulator considers a configuration change an error, since it would result in an abrupt reorganization of the links during the motion segment.) For such situations the simulator offers the following configuration management commands:

**get-solution <agent>**

prints out the currently used configuration index. This will give a value between 0 and 7 if the automatic selection method is being used.

The following command:

**get-valid-solutions <agent>**

prints the indices of all valid configurations for the current positions.

The following command:

**freeze-solution <agent>**

is equivalent to using 'set-solution' with the value obtained by using 'get-solution'.

The simulator also performs collision testing while moving the robots in the workspace. The collision testing can be enabled/disabled with the following command:

**set-collision-check <flag>**

If the flag is non-zero, collision checking is turned on, otherwise it is turned off. Initially the collision checking is enabled.

### 4.1.9 Position reporting

Position reporting is handled with the following commands:

**get-position <agent>**
**get-angles <agent>**
**record-angles <agent> <joint vector name>**

These commands write a line of the following format to the report stream:

**position of agent <agent> in line [label:NNN|NNN] --- <pos>**

These commands behave differently based on the operating mode of the simulator. If the simulator is in simulation only mode, then the manipulator model's joint angles or coordinates are reported. If the simulator is connected with a real robot manipulator then the robot hardware is queried for the actual joint angles, the simulator's model is updated with the reported angles, and these angles are printed out. This way the usage of these commands will synchronize the simulator's model with the actual manipulator. For the line numbering convention in the report stream see the explanation in section 4.1.2 on error reporting. The 'get-position' command prints the position in world (rectangular) coordinates, while the 'get-angles' command's output is in joint angles (degrees). The 'record-angles' command is similar to the 'get-angles' command, but it also records the angles in a coordinate vector which will be associated with the symbol specified in the command line. Coordinate vectors can be used as parameters for drive commands, and they are generally useful for recording important locations in the robot's workspace. The coordinate vector data base can be saved and restored with the following two commands:

**save-positions <filename>**
**load-positions <filename>**

As an additional safety measure against inadvertently losing important data, the simulator automatically saves the current coordinate vector set into the last used file if either a save or a load command was executed previously.

### 4.1.10 Grasping

The simulator's grasping operations are based on grasping attributes associated with each object in the workspace. These are the grasping coordinates (in an object relative coordinate frame) and the grasping opening used to establish contact with the

object. The grasping attributes are best specified immediately after object creation, when its location is still known, and then the grasping coordinates will be transformed any time the object is moved.

The following command establishes the grasping point and hand orientation in an object-relative coordinate frame:

**define-grasping-point <object> <x> <y> <z> <roll> <pitch> <yaw>**

The following command establishes the hand opening which is used to grasp the object. The hand is CLOSED to the distance specified upon grasping:

**define-grasping-opening <object> <distance>**

The following command is the EXTRA hand opening above the value specified above when the hand is moving in to grasp the object. This is a global value, but may be overwritten for an individual object by using:

**define-approach-opening <object> <distance>**

The default approach opening (= grasping opening + default gap) can be overwritten for individual objects using the above call.

The following command moves the hand to the grasping point of the object and opens it to the approach opening (defined using either the default gap or the individual approach opening commands). It will give an error message if the hand already holds an object. This command uses the 'find-path' command's method to get to the desired point.

**move-to-grasp <agent> <object>**

The following command grasps the selected object:

**grasp <agent> <object>**

If the hand is not empty, an error message is generated. Otherwise the hand closes to the grasping opening associated with the object, and in the simulation's data base a temporary link is set up between the object and the last link of the robot manipulator.

The following command releases the selected object:

**release <agent> <object>**

This is the opposite of grasp. It gives an error message if the hand is not holding the specified object. NOTE: the simulator does not model effects like gravity, etc. If an

object is released in the "air" it will stay there in the simulated environment, but of course it will drop in the real world, leading to inconsistencies between the world and its model.

### 4.1.11 General graphics setup

The following commands establish parameters for the viewing transformations:

**look-from <x> <y> <z>**
**look-at <x> <y> <z>**
**twist-camera <angle> <incremental>**

The twist-camera command is used only in the SGI implementation, where it rotates the camera around its axis. If the second parameter is given (its value is not important, it just serves as a place holder) then the twisting is done incrementally to the current camera angle, otherwise the angle parameter is interpreted as an absolute angle. In the HP-UX version this command has no effect, as is also true for the PC version.

The following command associates a color specification with the 'name' given to the color:

**define-color <name> <r> <g> <b>**

Red, green, and blue intensities range from 0.0 to 1.0.

The following command specifies the light source:

**light-source <x> <y> <z> <color> <ambient>**

This command is not needed for wire frame display, only for the other types. The ambient parameter just serves as a place holder, its value is not important. If it is present then the light source is ambient, otherwise it is directional.

The following command defines the graphics display option to be used:

**display-type wireframe|hidden|solid|shade**

Not all implementations support all display modes, if an unsupported mode is selected, the command is silently ignored.

### 4.1.12 Physical manipulator control

The following command enables sending commands to the real robots in the system:

**enable-execute <agent1> <agent2>**

**disable-execution**

The execution will be enabled for those manipulators only which are listed in this command. Their order must match the order in which the low-level interface function expects the coordinate vectors. The second command is the opposite of the first - only simulated execution afterwards. The system starts up in this state.

### 4.1.13 General

The following additional simulator commands are used for loading files, setting flags, and exiting the program:

**load <filename>** Takes commands from the specified file. Returns when end of file or the 'exit' (or 'end') command is encountered. Loads may be nested.

**set-echo <flag>** If 'flag' is non-zero then all subsequent loads will echo the contents of the command file as it is being processed.

**exit** or **end** If given at the simulator prompt, exits the simulator program returning to the DOS prompt. If given in a command file, returns to the simulator prompt.

**abort** If given at the simulator prompt, exits the simulator program returning to the DOS prompt in the same manner as the 'exit' or 'end' command. If given in a command file, exits the simulator program returning to the DOS prompt.

## 4.2 Simulation Example

Figure 3 shows a simulated arc-welding workcell for loading, positioning, and welding a four-part assembly. Listing 3 shows the command sequence used to set up the workcell of Figure 3. Figure 3 is a screen dump with the same resolution as viewed on the monitor. Figure 4 shows the same workcell printed with high resolution on an HP Laserjet printer.

Referring to Figure 3, the workcell consists of two robots, a positioner (turntable), two work tables, a welding torch and torch holder, and four parts to be welded into an assembly. The robot on the left-hand side sequentially loads the parts onto the positioner which positions the parts (in some cases, rotates the part) for welding by the robot on the right-hand side of the cell. A portion of the weld command sequence used for loading the first two parts of the assembly and performing the first quadrant weld is shown in Listing 4. Figure 5 shows the first two parts of the assembly being welded. Figure 6 is a closeup view of the first weld. The completed assembly is shown in Figure 7. On the computer screen the entire welding operation is performed with high speed animation in color.

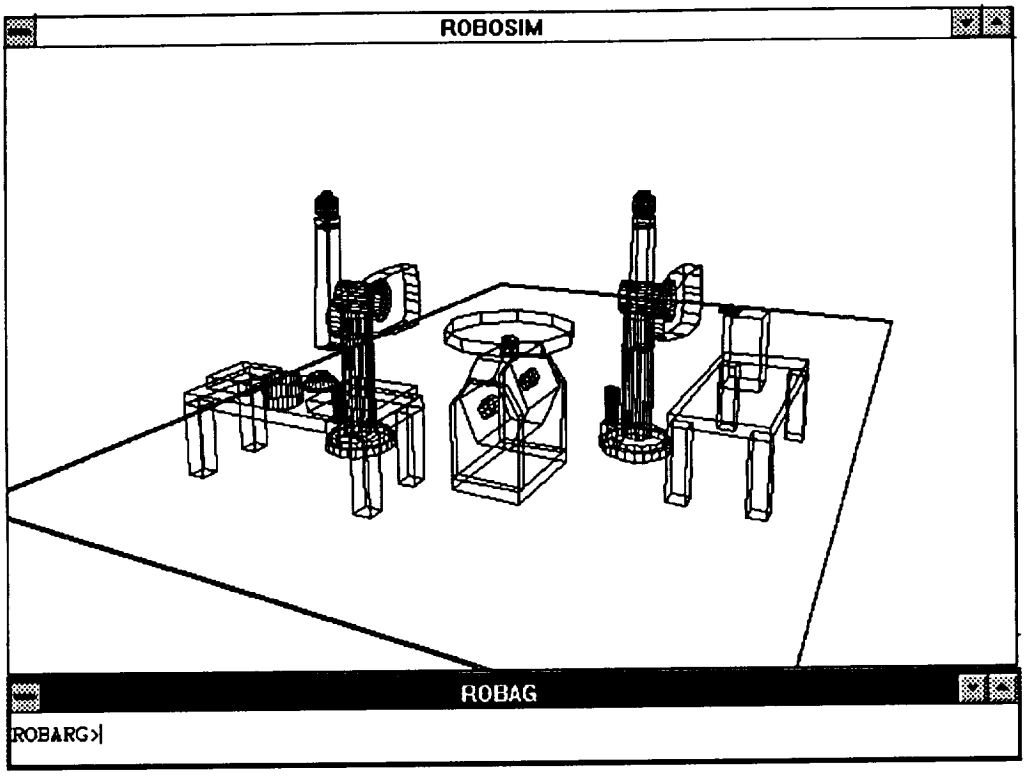


Figure 3. Simulated arc-welding workcell.

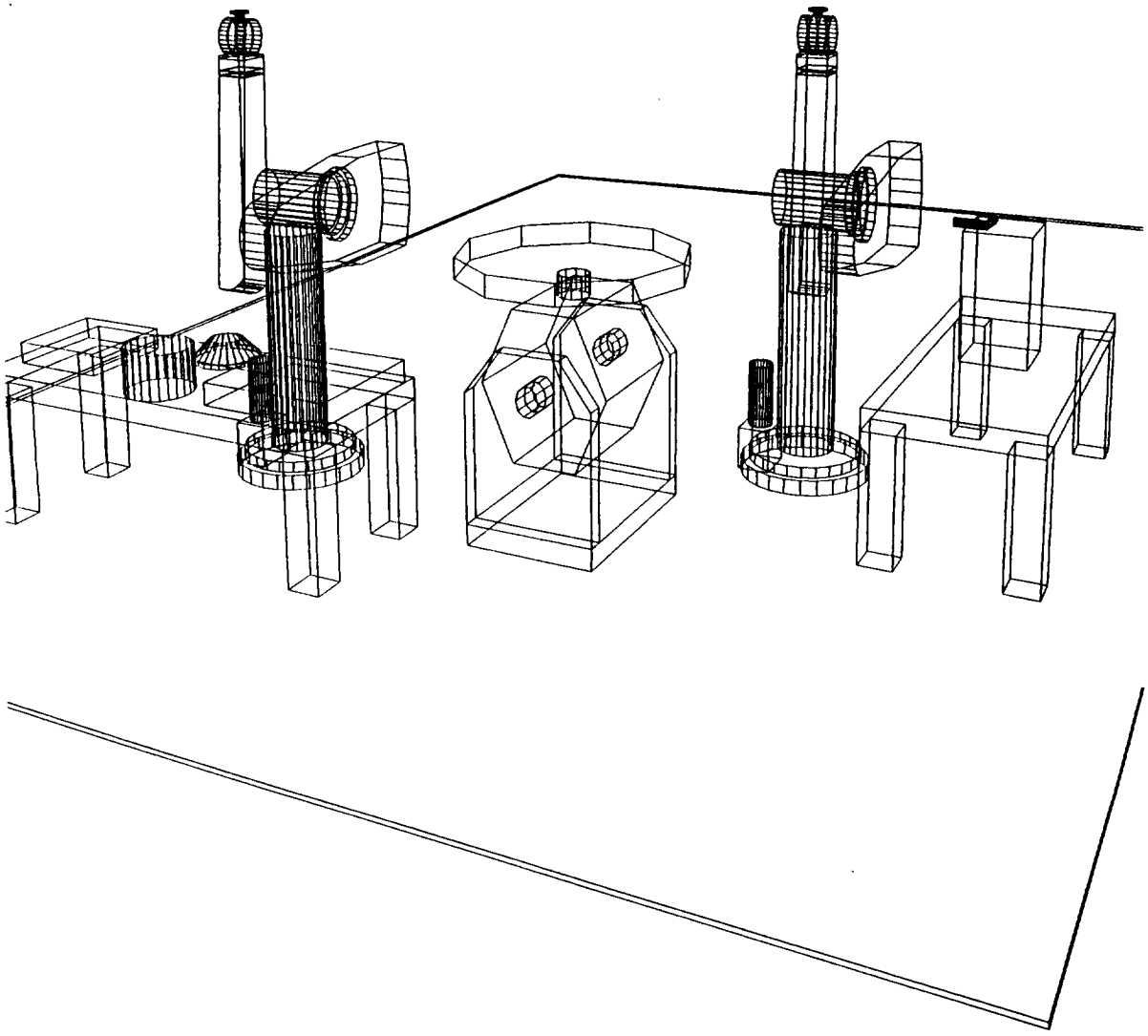

Figure 4. Laser printout of welding workcell.

```
;*********************
;
;        SETUP.CMD
;*********************
;
define-color red 1.00 0.00 0.00 ;                    COLOR DEFINITIONS
define-color green 0.00 1.00 0.00
define-color blue 0.00 0.00 1.00
define-color purple 1.00 0.00 1.00
define-color cyan 0.00 1.00 1.00
define-color yellow 1.00 1.00 0.00
define-color white 1.00 1.00 1.00
define-color lt-grey 0.75 0.75 0.75
look-at 0 0 500 ;                                    VIEW COMMANDS
look-from -2000 4000 1500
set-collision-check 0 ;                              SET COLLISION OFF
minimal-step 4 ;                                     SET INITIAL SPEED
make-object floor box 4000 6000 10 lt-grey ;         FLOOR
translate-object floor x=350 y=-1500 z=-10
make-agent p position cyan ;                         POSITIONER
make-agent r1 puma560 purple ;                       ROBOT - R1
rotate-object r1 z=90
translate-object r1 x=-500 y=-650
make-agent r2 puma560 green ;                        ROBOT - R2
rotate-object r2 z=90
translate-object r2 x=1000
make-robosim-object table1 table7.dat table1 white ; TABLE1
translate-object table1 x=1000 y=500
make-robosim-object table2 table7.dat table1 blue ;  TABLE2
rotate-object table2 z=90
translate-object table2 x=-1100 y=-650 ;             TORCH HOLDER
make-object holder box 200 200 300
make-object box1 box 400 400 50 green ;              BOX1 OF ASSEMBLY
translate-object box1 x=600 y=500 z=375
define-grasping-point box1 0 0 25 180 0 0
make-object box2 box 282 282 50 yellow ;             BOX2 OF ASSEMBLY
translate-object box2 x=1400 y=500 z=375
define-grasping-point box2 0 0 25 180 0 0
make-object pipe cylinder 100 100 cyan ;             CYLINDRICAL PART
translate-object pipe x=1000 y=650 z=400
define-grasping-point pipe 0 0 50 180 0 0
make-robosim-object top top.dat top white ;          TOP OF ASSEMBLY
translate-object top x=1000 y=350 z=375
define-grasping-point top 0 0 26 180 0 0
make-robosim-object torch torch.dat torch red ;      WELDING TORCH
define-grasping-point torch 0 0 -50 0 0 -180
rotate-object torch y=-90
```

```
translate-object torch x=-1000 y=-650 z=650
end
```

Listing 3. Setup Commands for Arc-Welding Workcell Shown in Figure 3

```
; ****************************************
;GET WELDING TORCH AND POSITION POSITIONER
; ****************************************
move-to-grasp r1 torch , drive p -90 0 0 0 0 0
grasp r1 torch
drive r1 0 0 0 0 0 0
; ****************************************
;LOAD BOX1 OF BOTTOM PART OF ASSEMBLY
; ****************************************
move-to-grasp r2 box1
grasp r2 box1
drive r2 0 0 0 0 0 0
move-inter r2 330 0 350 -180 90 0
release r2 box1
drive r2 0 0 0 0 0 0
; ****************************************
;LOAD BOX2 OF BOTTOM PART OF ASSEMBLY
;****************************************
move-to-grasp r2 box2
grasp r2 box2
drive r2 0 0 0 0 0 0
move-inter r2 380 0 350 -90 45 90
release r2 box2
drive r2 0 0 0 0 0 0
; ****************************************
;LINK BOX1 AND BOX2 INTO "BOTTOM" PART
; ****************************************
link-objects bottom box1 box2
; ****************************************
;POSITION BOTTOM PARTS FOR WELDING
; ****************************************
grasp p bottom
drive p 45 0 0 0 0 0
drive r1 -34.5 -30 125 0 62 0
; ****************************************
;WELD FIRST QUADRANT
; ****************************************
look-at -153 0 819
look-from -153 1000 819
minimal-step 1
```

```
move-straight r1 -154.26 -151.89 812.07 -180 -21 41
look-at 0 0 500
look-from -2000 4000 1500
minimal-step 10
drive r1 0 0 0 0 0 0
.
.
.
```

Listing 4. Weld Command Sequence for First Quadrant Weld on Bottom Assembly

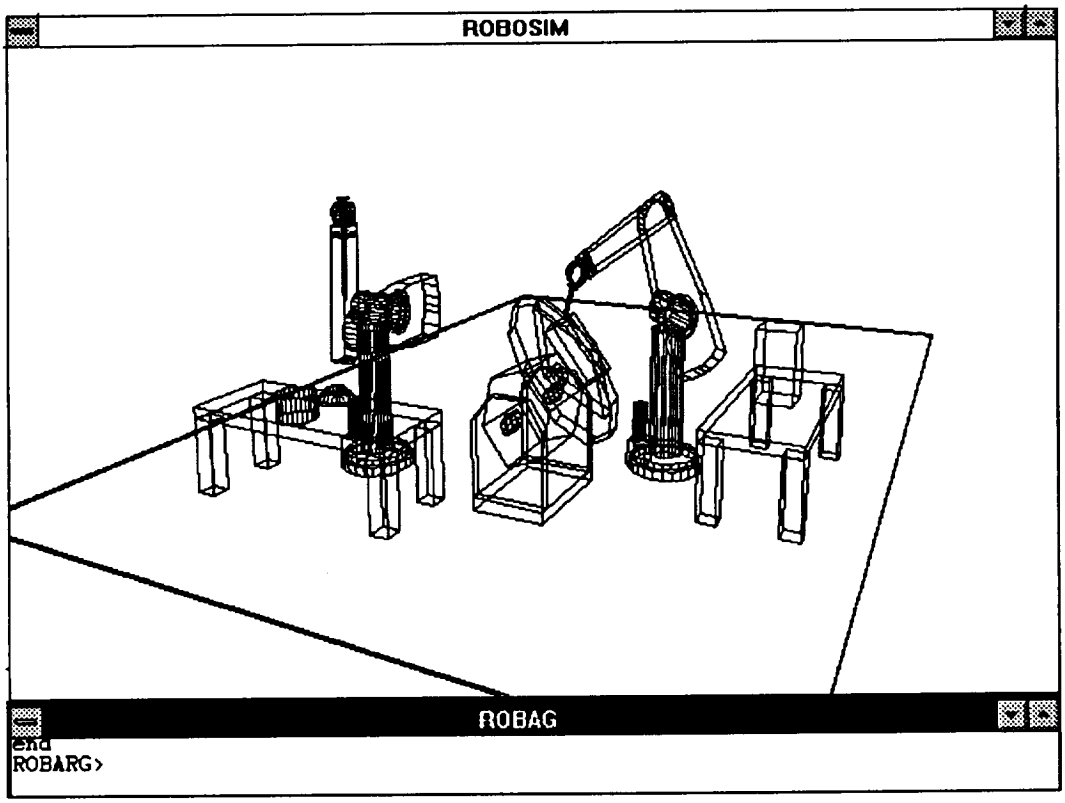


Figure 5. Welding first two parts of assembly.

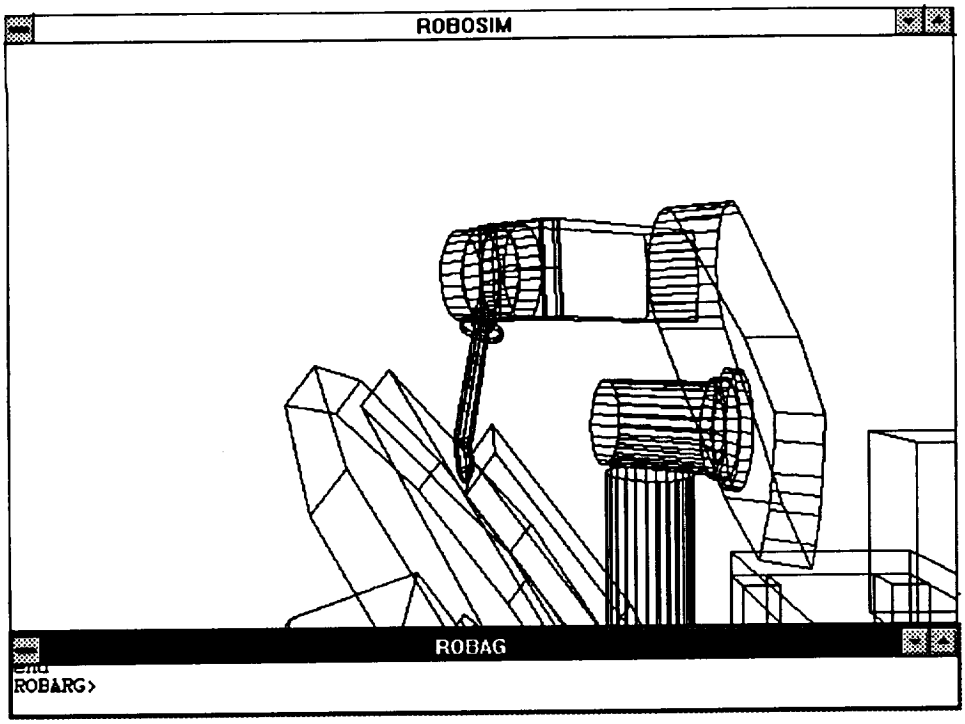


Figure 6. Closeup of welding operation.

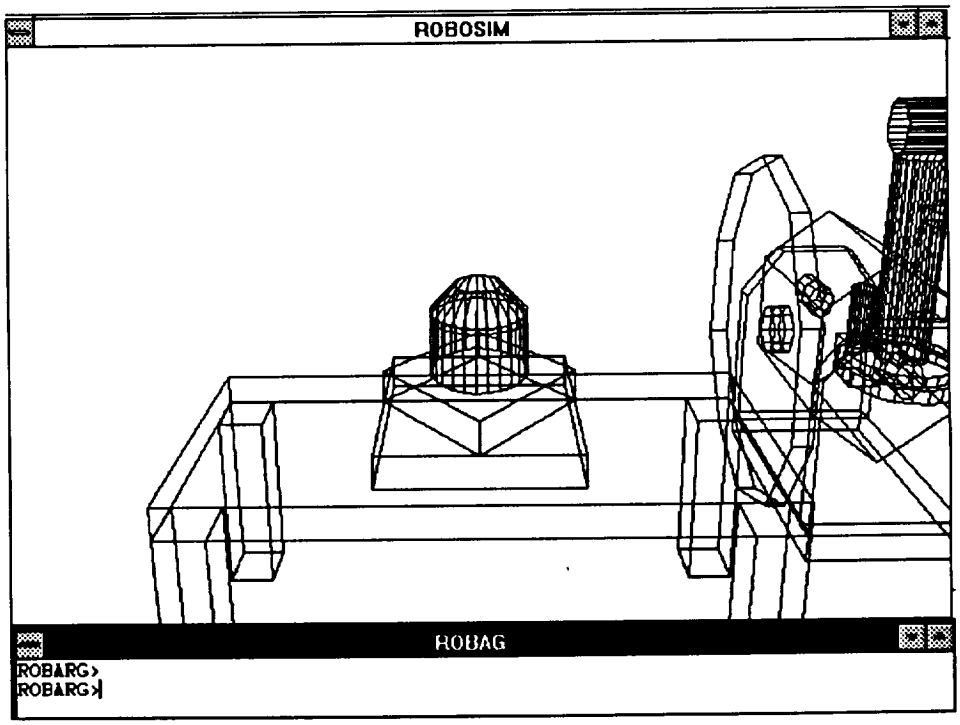


Figure 7. Completed weld.

## 5.0 ADVANCED FEATURES

Although the basic modeling and simulation environment described above perform quite satisfactorily as a robot modeling tool, ROBOSIM's capabilities were greatly enhanced with the completion of various extensions to the basic package. Some of these extensions use heuristic, rule-based programming techniques. The extensions include:

- *Composite objects:* The ROBOSIM simulation environment also supports the linking of separate objects into a so-called composite object which can be operated on as a unit while preserving the separate accessibility of its components. A good example of this is a drawer with various objects in the drawer. During the course of the simulation a manipulator may have to pull out the drawer (the drawer and its contents have to be treated as a unit) and then pick up a single object from the drawer (now a part of the composite object has to be accessed individually).

- *Configuration management:* When a manipulator arm is programmed using world coordinates it is typical to have several valid solutions (sets of joint coordinates) by which the manipulator can reach the desired position. For example, in the case of manipulators similar to the Unimation PUMA 560 robot this manifests in left or right handed and elbow up or elbow down configurations for the arm. ROBOSIM permits the user to choose any configuration and stay with it for the duration of the simulation. However, this approach is not optimal when the manipulator has to move distances comparable to the limits of its envelope. For such cases the simulator provides an automatic configuration selection mechanism which is based on the following heuristics:

  - √ *For small displacements stay with the current configuration:* The rationale behind this rule is that small displacements typically mean local manipulation on some object.

  - √ *The configuration **must not** change during straight-line motion segments.*

  - √ *For large displacements use the configuration which gives the most freedom at the target position:* The configuration giving the most freedom is defined as the solution which has all of its joint coordinates as far from the limits of the manipulator as possible. The rationale behind this heuristic is that the manipulator is expected to perform local manipulations on the objects near its new location for which it needs as much freedom as possible.

The above heuristic rules have been observed to perform reasonably well in a number of simulations. Only in a few cases was it necessary to manually override the configuration selected by them.

- *Collision detection:* ROBOSIM also provides a way to check for collisions during a simulation run. The collision detection is based on the detection of intersections of solid object (passive objects or manipulator links) bounding polygons. For efficiency reasons the simulator maintains a rectangular bounding box for every object and invokes the more complex polygon-based collision detection method only if the bounding boxes intersect.

- *Collision avoidance:* The simulator also provides a heuristic path planning algorithm which is capable of recovering from collision situations. The collision avoidance is based on heuristic rules describing actions to try in various collision situations. Some of these rules are generic, others are manipulator-specific. An example generic rule is the "minimal volume rule" which is usable for large displacements of the end effector. It specifies a path to reach the desired target which includes a midpoint where the arm is folded in a way which minimizes its reach. Users can attach other heuristic rules specific to their manipulator models. The heuristic collision avoidance algorithm works as follows:

  (1) Verify that the target position is collision free. If not, the arm cannot reach the target.

  (2) Attempt to reach the target position from the current position using the specified path synthesis method. (Most of the time this is just a simple joint-interpolated motion.)

  (3) If a collision is detected between the starting and end points do the following:

    ✓ Determine the collision situation (objects involved, direction of movement, etc.).

    ✓ Match the capabilities of the available heuristic rules against the collision situation.

    ✓ Invoke all matching rules in sequence. The rule will determine an intermediate point which, when used to divide the current motion segment will hopefully avoid the collision situation.

    ✓ Recursively repeat the planning process for both segments of the now divided original motion segment.

Additional checks are built into the planning algorithm to make sure that it does not get into an infinite recursion. Additionally, the algorithm stores already tried motion segments to speed up the search.

- *Interface to control real robots:* The simulation environment is also capable of generating command sequences for real robot controllers. In this mode only those motion commands are output which have been verified with the built-in collision checking to be safe. Currently only the PUMA 560 robot with the Unimation controller running the VAL II robot control system [5] is supported, but additional output modules can be added.

A simple simulation for demonstrating the features of configuration management, collision detection, and collision avoidance is shown in Figure 8. This is the same simulation as presented previously in Figure 2. It is repeated here for convenience. The SETUP.CMD file used to create the simulation was presented previously in Listing 2, page 9. The simulation consists of a PUMA 560 robot (named 'r'), a floor, an object (named 'xx'), and a ceiling structure (named 'top').

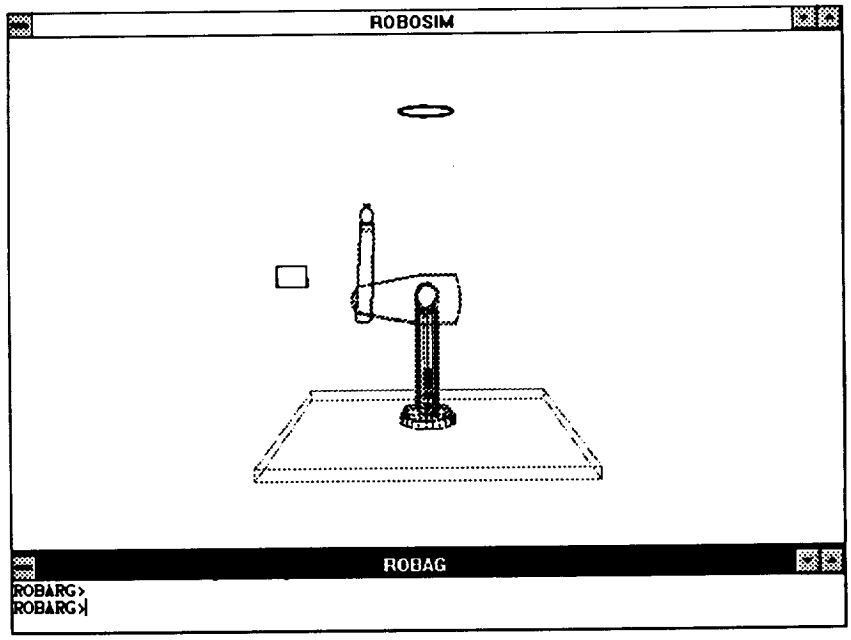


Figure 8. Simulation of object removal/insertion for demonstration of configuration management, collision detection, and collision avoidance (object on left side).

The motion command file is given in Listing 5, and the referenced command files MOVEFORW.CMD and MOVEBACK.CMD are given in Listings 6 and 7 respectively. The objective of the motion demo is to remove the object xx from its initial location at x = 900, y = 300, and z = 800 and reinsert it at a new location x = 900, y = -300, and z = 800. This is done first in the left handed, elbow down configuration of the robot (solution #0) and then in the left handed, elbow up configuration (solution #3). The object xx is them relocated to the right side of the simulation (see Figure 9). and the removal/insertion

operation is repeated with the robot in its right handed, elbow up configuration (solution #4) and finally in its right handed, elbow down configuration (solution #7).

```
;*********************
;       MOTION.CMD
;*********************
set-solution r 0;              LEFT HANDED, ELBOW DOWN CONFIGURATION
load moveback.cmd;             REMOVE OBJECT AND REINSERT AT Y = -300
load moveforw.cmd;             REMOVE OBJECT AND REINSERT AT Y = 300
set-solution r 3;              LEFT HANDED, ELBOW UP CONFIGURATION
load moveback.cmd
load moveforw.cmd
rotate-object xx z=180;        MOVE OBJECT TO RIGHT SIDE OF SIMULATION
translate-object xx x=200
set-solution r 4;              RIGHT HANDED, ELBOW UP CONFIGURATION
load moveforw.cmd
load moveback.cmd
set-solution r 7;              RIGHT HANDED, ELBOW DOWN CONFIGURATION
load moveforw.cmd
load moveback.cmd
translate-object xx x=-200;    RETURN OBJECT TO LEFT SIDE OF SIMULATION
rotate-object xx z=180
end
```

Listing 5. Motion commands for object removal/insertion demonstration, Figure 8.

```
;*********************
;       MOVEFORW.CMD
;*********************
move-to-grasp r xx
get-solution r
grasp r xx
translate-agent r x=-200
translate-agent r y=600
translate-agent r x=200
release r xx
drive-find r 0 0 0 0 0 0
end
```

Listing 6. Move forward command sequence, Figure 8.

```
;*********************
;       MOVEBACK.CMD
;*********************
move-to-grasp r xx
```

```
get-solution r
graspr xx
translate-agent r x=-200
translate-agent r y=-600
translate-agent r x=200
release r xx
drive-find r 0 0 0 0 0 0
end
```

Listing 7. Move back motion command sequence, Figure 8.

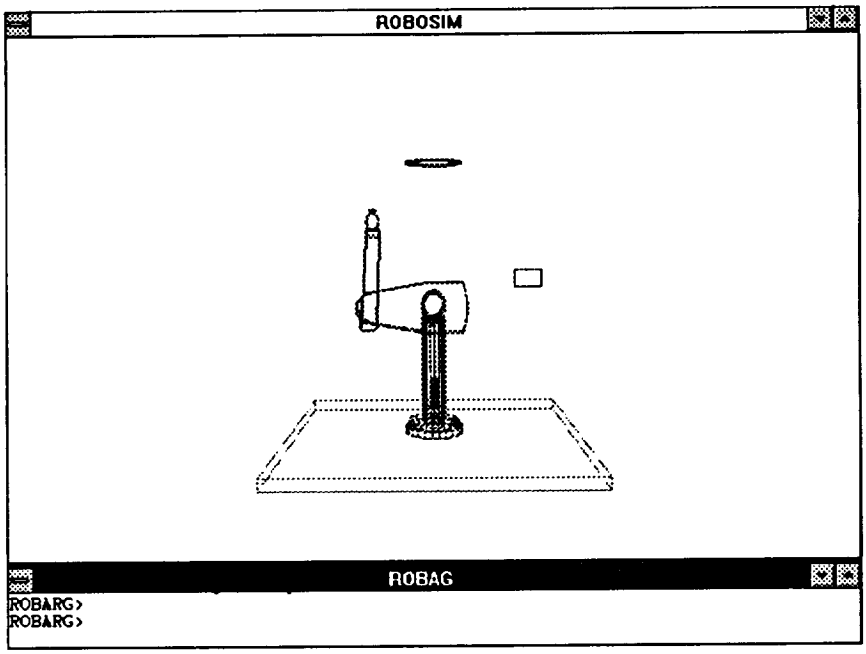


Figure 9. Simulation of object removal/insertion for demonstration of configuration management, collision detection, and collision avoidance (object on right side and ceiling lowered by 250 units to cause collision with robot when fully extended).

Referring to the motion commands sequence (Listing 5), the robot configuration is first set to the left handed, elbow down solution (#0). The MOVEBACK.CMD sequence is then loaded followed by the MOVEFORW.CMD commands. Referring to Listing 6, the *move-to-grasp* command is used to position the robot arm for grasping the object. As explained previously, the *move-to-grasp* command uses the *find-path* command's method to get to the desired point. Thus the collision avoidance heuristics are called upon to determine a collision free path if a collision is detected. With the ceiling at its normal height, as shown in Figure 8, no collisions will be encountered in any of the four arm configurations tested. Once the robot arm is in position to grasp the object, it is grasped and the *translate* command is used to move the object in straight lines for the removal and insertion operations. The object is first translated along a straight line in the x-direction by

-200 units. It is then translated along a straight line in the y-direction by 600 units, and then reinserted by translating along the x-direction by 200 units. This operation might correspond to the removal of a spare module from a rack and insertion of the module into a system being repaired. The robot arm is shown at the instant of grasping the object for removal in Figure 10 and at the final insertion position in Figure 11. Both figures are for the elbow down configuration.

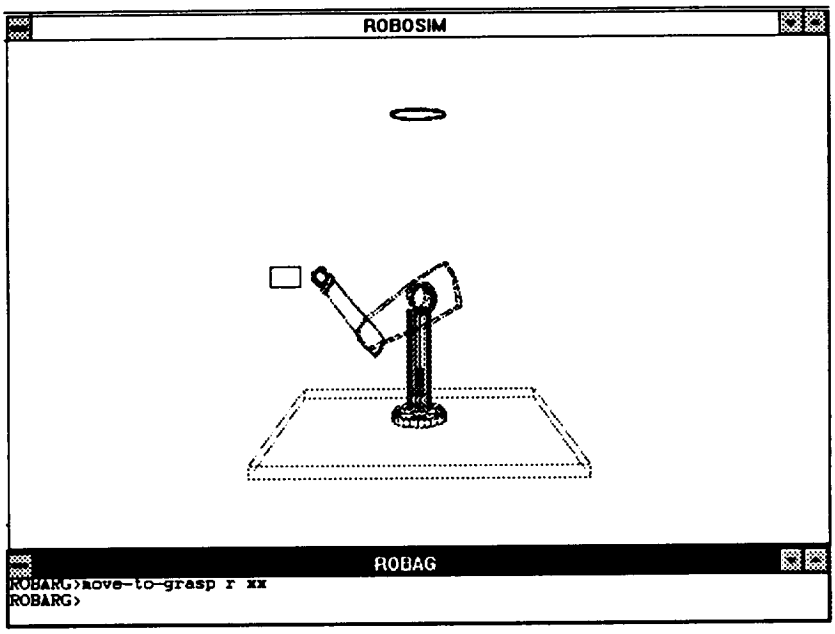


Figure 10. Robot in position to grasp object for removal, solution index 0.

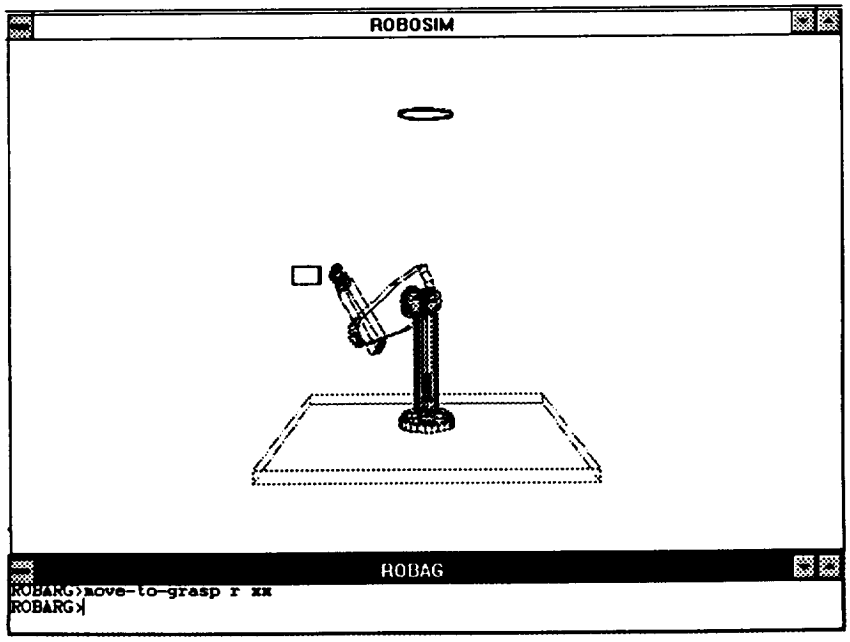


Figure 11. Robot at module insertion position, solution index 0.

Figures 12 and 13 show the module removal and insertion operations in the elbow up configuration. The solution index is #3.

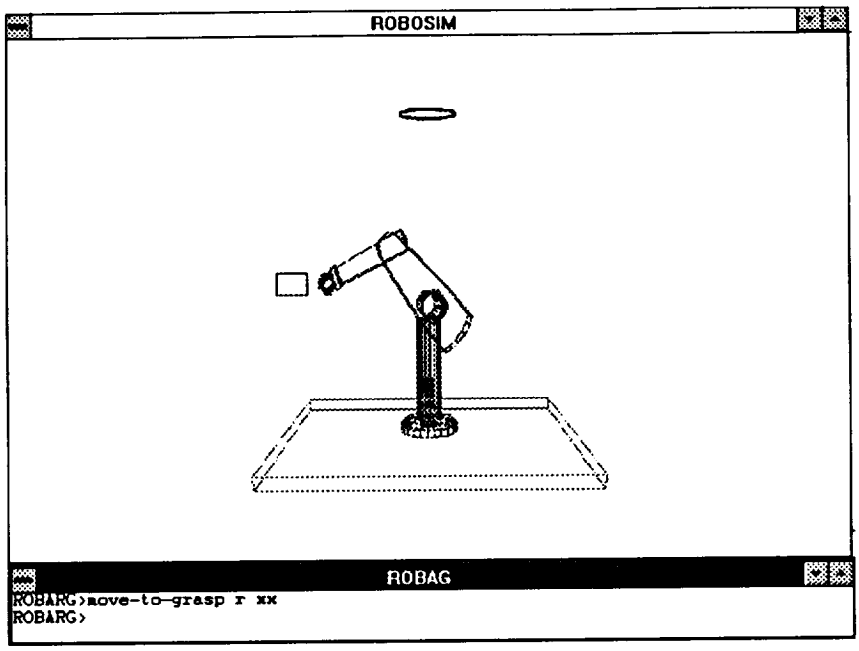


Figure 12. Robot in position to grasp object for removal, solution index 3.

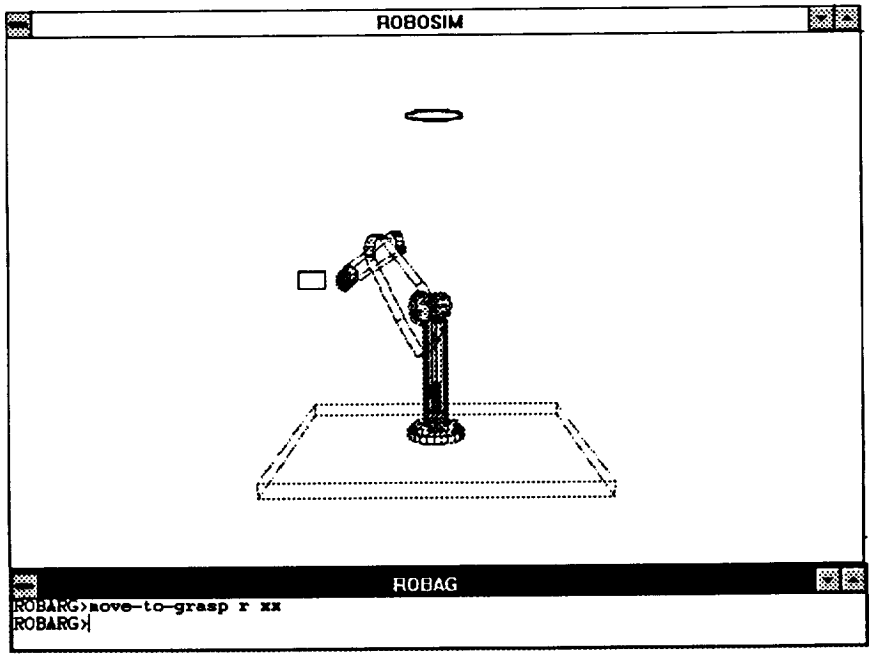


Figure 13. Robot at module insertion position, solution index 3.

Referring to Listing 5, the object is next moved to the right side of the simulation and the module removal/insertion operations are repeated first in the right hand, elbow up

configuration (solution index #4) and then in the right hand, elbow down configuration (solution index #7). Figures 14 and 15 show the module removal/insertion operations in the elbow up configuration, and Figures 16 and 17 show the operations in the elbow down configuration.

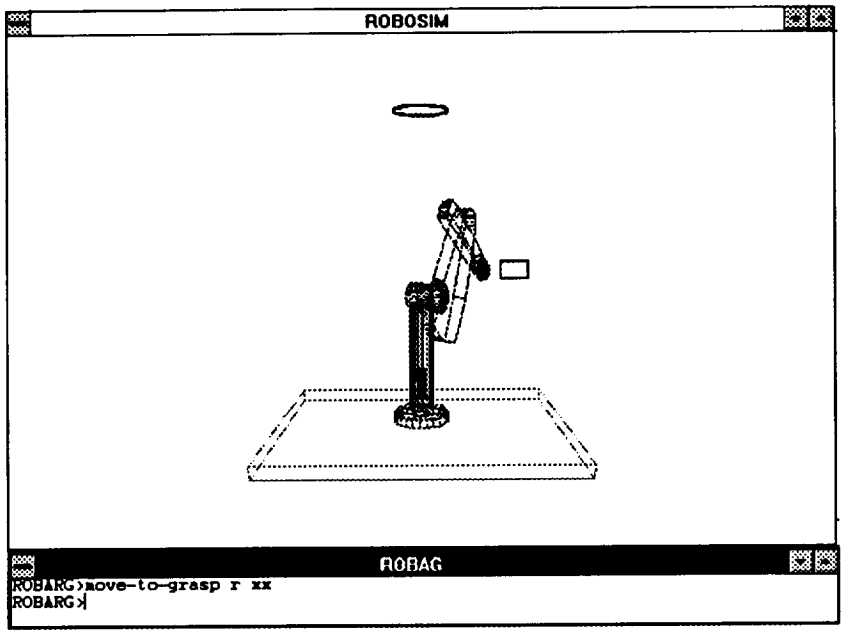


Figure 14. Robot in position to grasp object for removal, solution index 4.

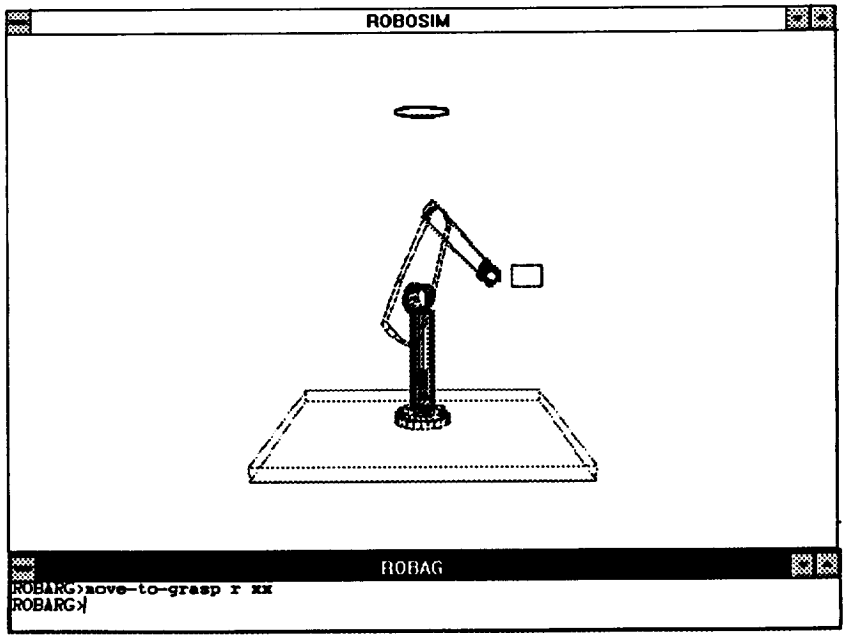


Figure 15. Robot at module insertion position, solution index 4.

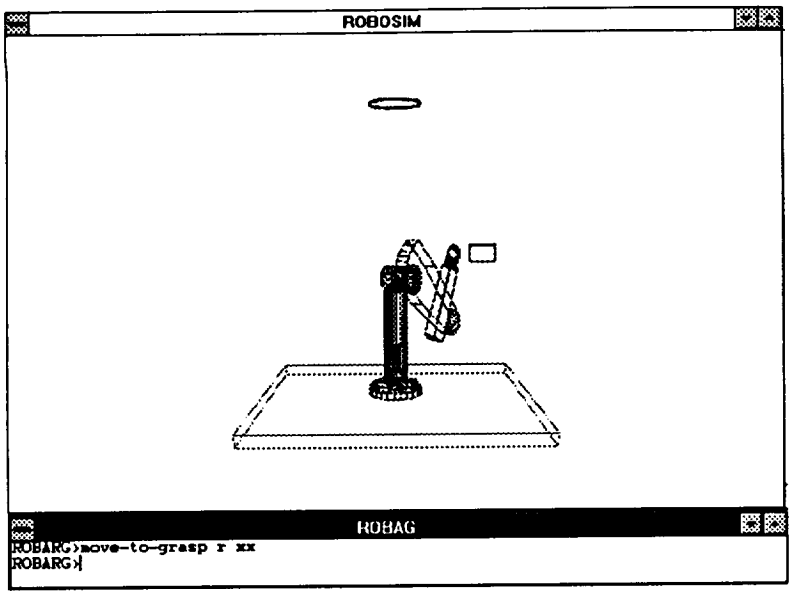


Figure 16. Robot in position to grasp object for removal, solution index 7.

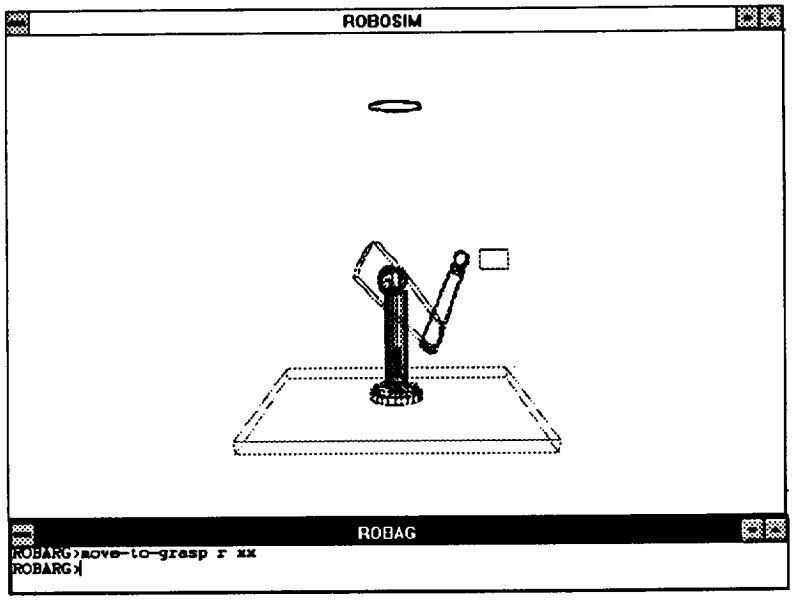


Figure 17. Robot at module insertion position, solution index 7.

As previously observed, the *move-to-grasp* command is used to position the robot arm for grasping the object, and the *move-to-grasp* command uses the *find-path* command to get to the desired point. The *find-path* heuristics first attempts to reach the target using joint-interpolated motion. For the elbow down configurations of Figures 16 and 17, this calls for moving from joint angles of 0 (home position shown in Figure 9) to joint angles on the order of -200 for joint 2 and 200 for joint 3. This results in the arm being nearly fully extended as it swings through the vertical position, as shown in Figure 18, on its way to the position shown in Figure 16. As can be seen in Figure 18, this does not result in a

collision with the top so long as it is at its normal position. However, if the ceiling is lowered by 250 units, a collision does occur, as shown in Figure 19, when the arm attempts to move from its home position to the grasp position for the solution index #7 configuration.

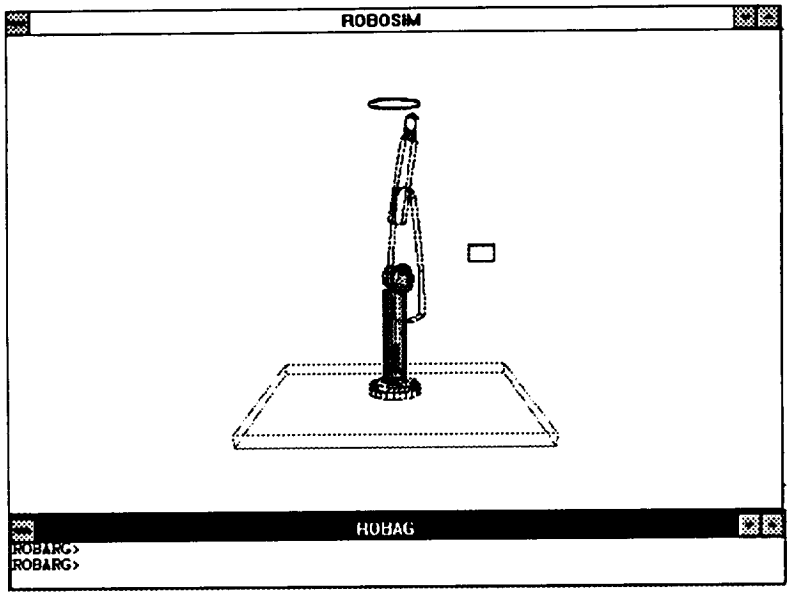


Figure 18. Robot arm fully extended as it swings from home position to grasping position.

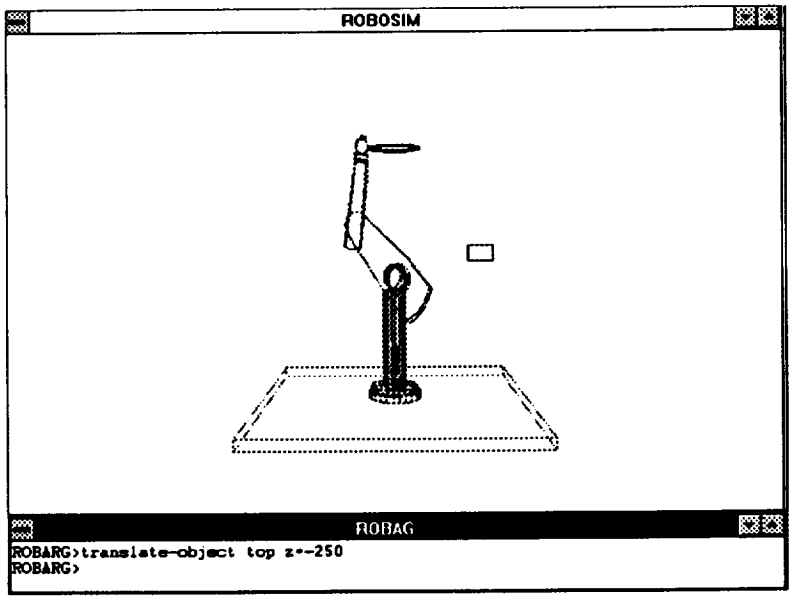


Figure 19. Collision occurs when ceiling is lowered, calling for new strategy from path planning heuristics.

The first approach tried by the path planning heuristics is to "fold up" the third joint as the second joint swings through its motion. In this case, this works as shown in Figure

20, so the heuristic continues on to the position of Figure 21 where simple joint-interpolated motion is tried again to reach the target position.

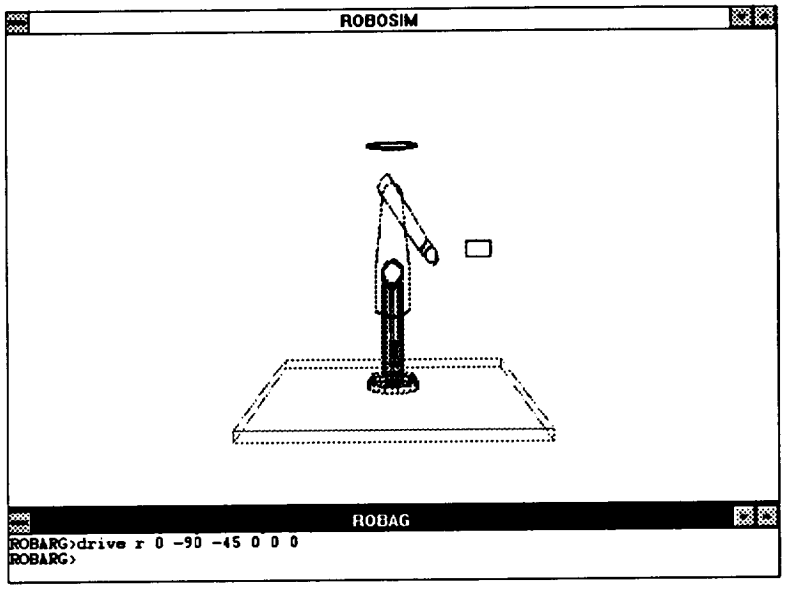


Figure 20. Path planning heuristics successfully avoids collision by "folding" third joint as second joint swings through its motion.

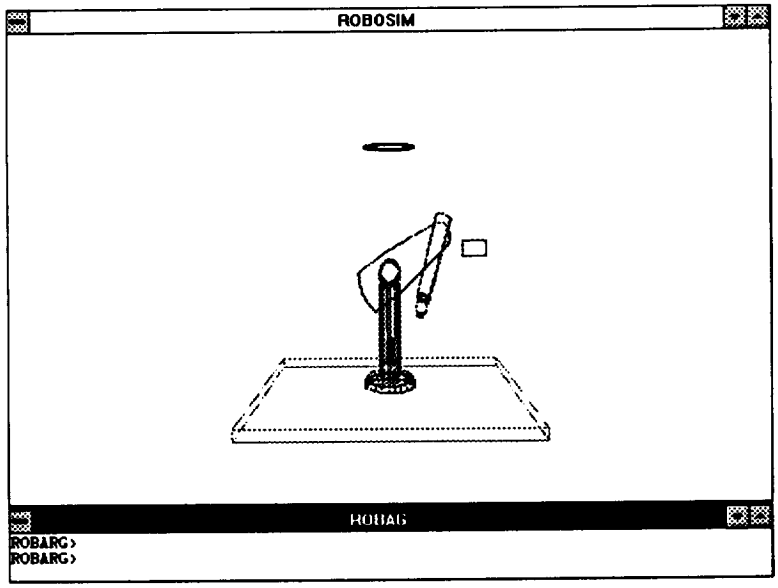


Figure 21. Having successfully avoided collision with lowered ceiling, path planning heuristics continue with attempt to reach target from intermediate position with simple joint-interpolated motion.

In attempting to move from the intermediate position shown in Figure 21 to the target position invoking joint-interpolated motion, a second collision occurs between the arm and the object as shown in Figure 22. The path planning heuristics successfully avoid

this collision by swinging the first joint temporarily beyond its target value (Figure 23) before approaching the final grasping position shown in Figure 24. In this case, two collisions were detected and avoided by the first approach attempted by the path planning heuristics in each case. For more difficult situations, the collision avoidance heuristics may try a number of approaches before finding one that works.

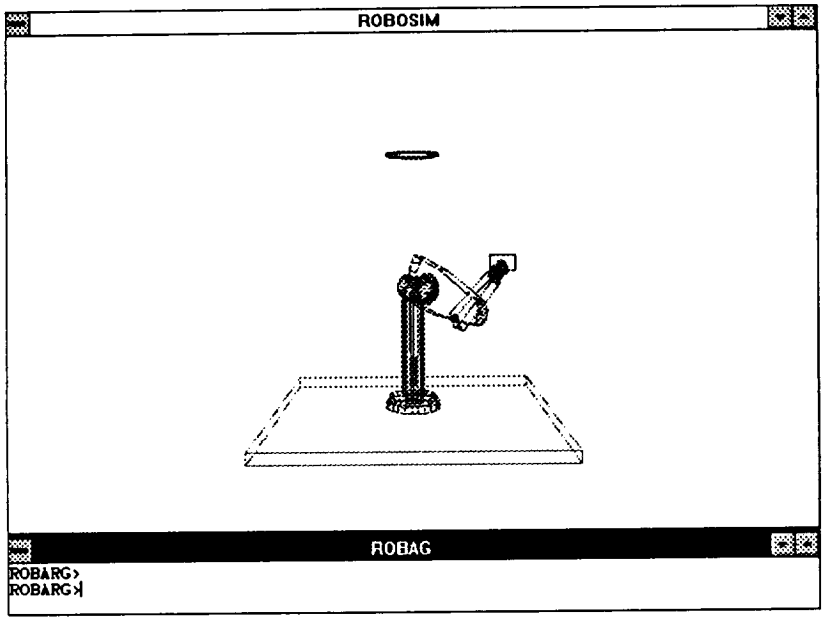


Figure 22. Second collision occurs when arm moves from intermediate position needed to avoid first collision by simple joint-interpolated motion.

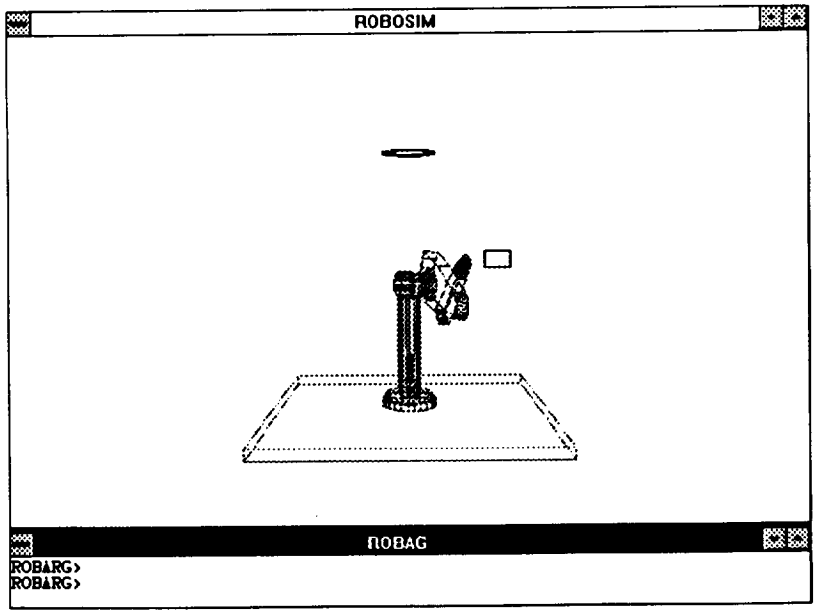


Figure 23. Path planning heuristics avoid second collision by temporarily swinging first joint beyond its target value.

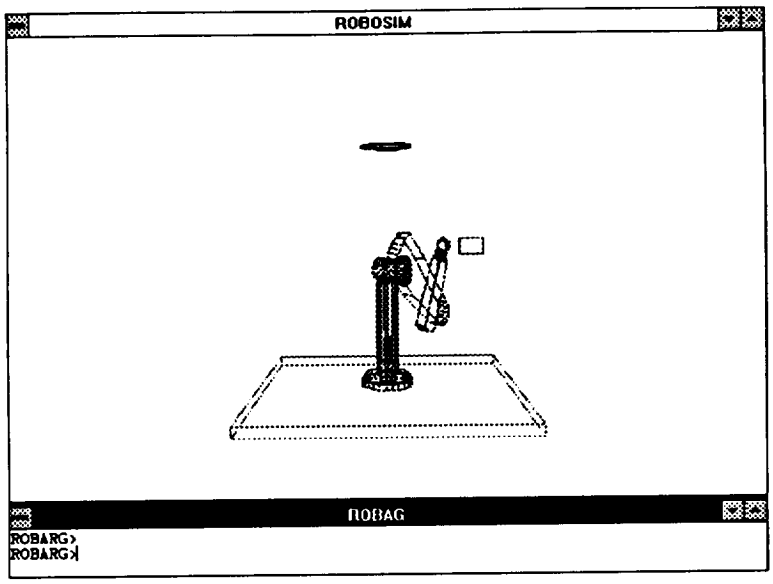


Figure 24. Final target position reached by avoiding two collisions.

## 6.0 APPLICATIONS

The ROBOSIM modeling and simulation package has been used both in teaching and research. It has been used in the teaching of courses on robotics, mechanisms, industrial control, and advanced automation at both the undergraduate and graduate levels [6,7]. ROBOSIM provides the students with a graphical means of visualizing objects and their relationships in 3-dimensional space. With the simulation system, students may actually graphically construct every manipulator they study in the classroom or in class assignments, and then they may operate the manipulator and use it to grasp, move, and release objects, and develop and test action plans for using the manipulators in practical settings. The cost of doing this with physical hardware would of course be quite prohibitive in terms of both monetary costs and time. Figure 25 shows students studying a robot servicing scenario on a space station concept. Figure 26 shows two PUMA 560 robots in a dual-robot, coordinated motion welding application. One robot holds the part while the other robot manipulates the welding torch. The motion is coordinated to maintain the joint in an instantaneous horizontal position at the time of welding with the torch vertical and welding speed constant. Figure 27 shows simulation models of Cincinnati Milacron T3 robots on a Hewlett Packard workstation.

ROBOSIM has also been used in a number of research projects sponsored by the Strategic Defense Initiative and by the National Aeronautics and Space Administration dealing with various uses of robots in space-related applications [8-11]. It has also been used in a research project sponsored by Boeing Aerospace, Huntsville, AL, which had the goal of building and integrated automatic diagnostics and robotics repair system prototype for Space Station Freedom [10]. Figure 28 demonstrates optimum downhand welding on the space shuttle main engine nozzle. Figure 29 shows a simulated saddle weld on the planned space station berthing port.

Figure 25. [Top] Students manipulating a robot servicing scenario on a space station concept. [Bottom] Space station concept with servicing robot (in red).

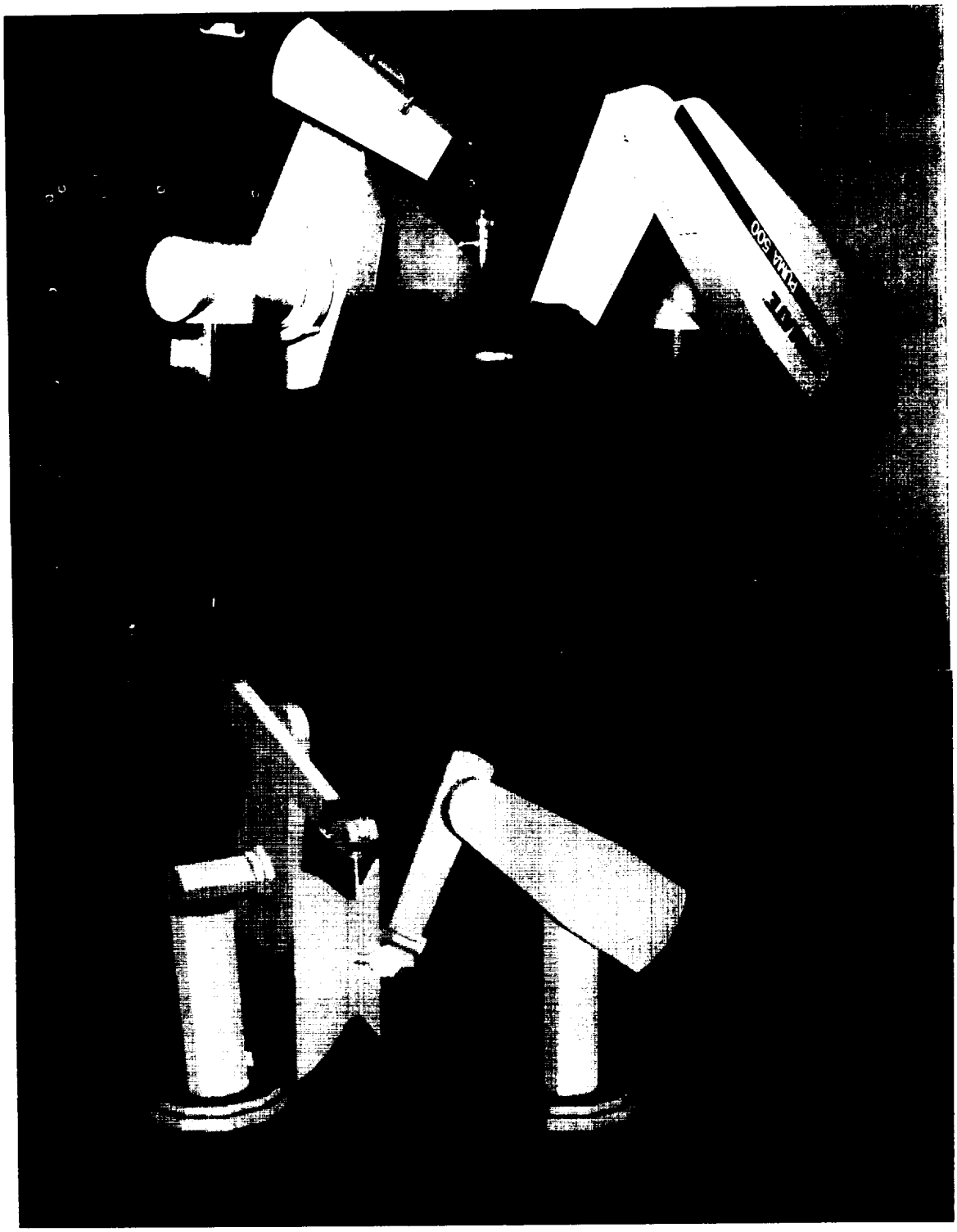


Figure 26. PUMA 560 robots in a dual-robot, coordinated motion welding application. [Top] Physical robots. [Bottom] Simulation.

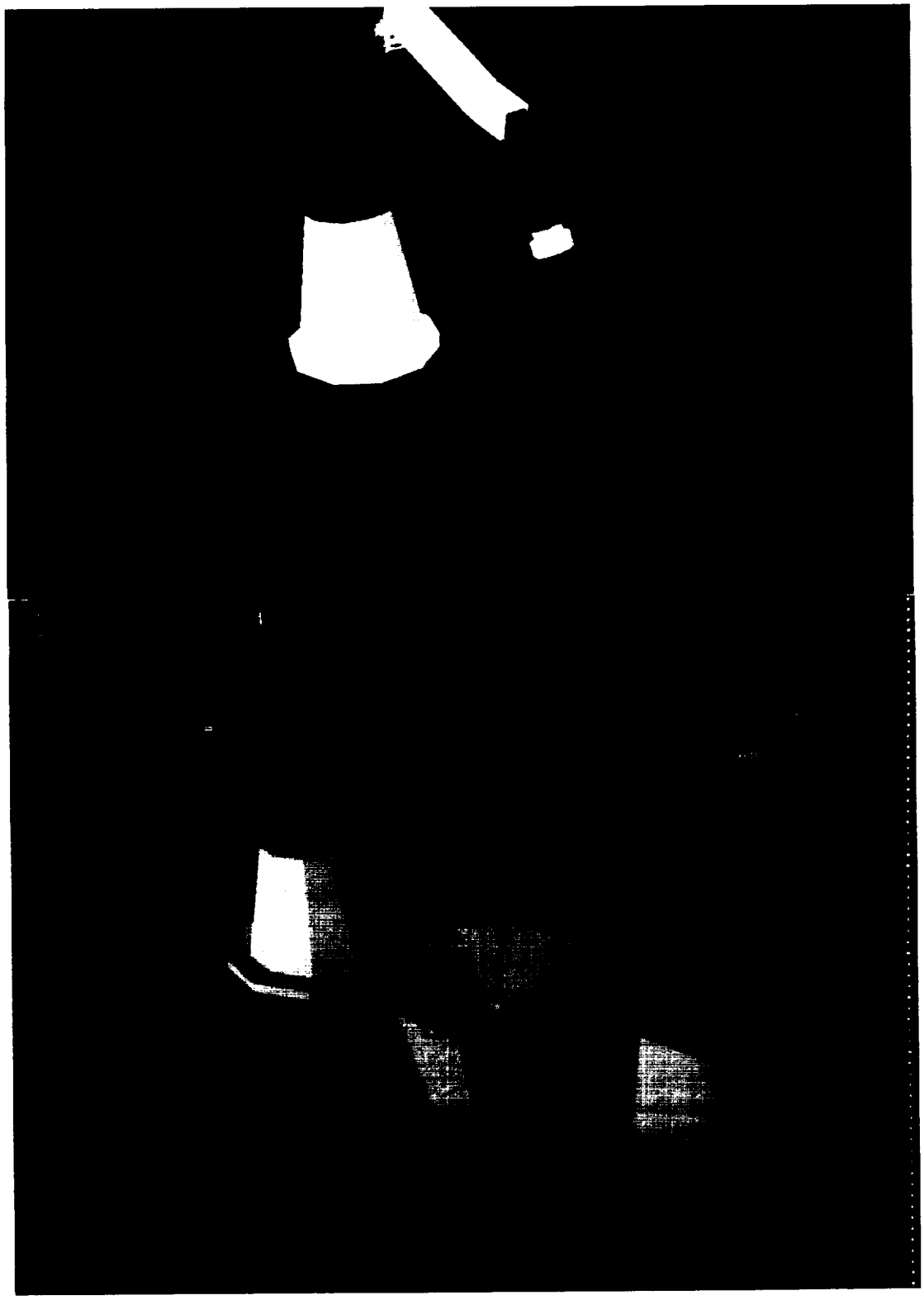

Figure 27. [Top] Cincinnati Milacron T3 robots. [Bottom] Construction of components.

Figure 28. Optimum downhand welding on the space shuttle main engine nozzle with coordinated motion of positioner and manipulator.

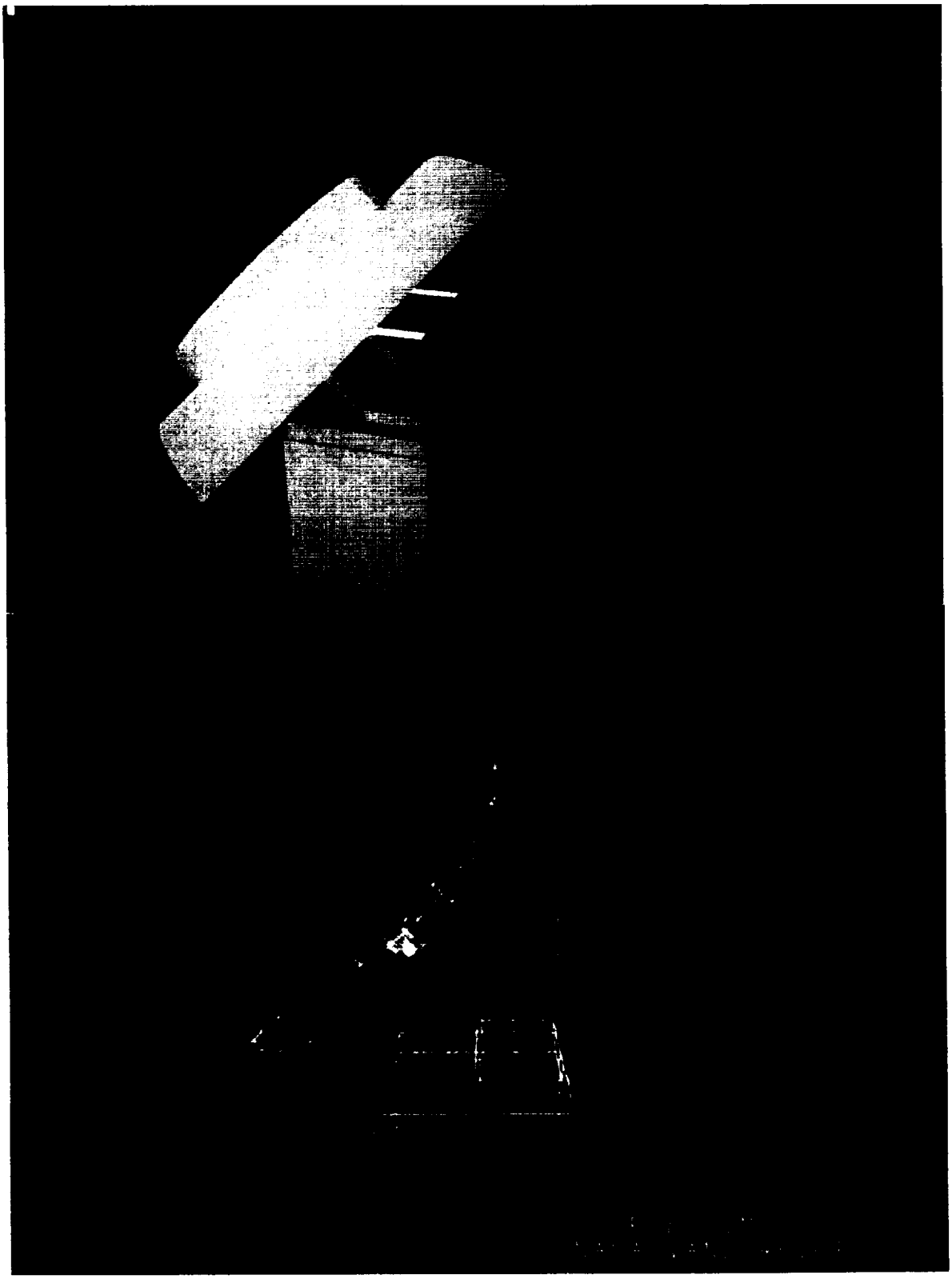

Figure 29. Simulated saddle weld on space station berthing port concept. [Top] Shaded modelling. [Bottom] Wireframe modelling.

# 7.0 CONCLUSIONS, FUTURE ENHANCEMENTS

The available experience with the ROBOSIM package suggests that it is a flexible and powerful tool for modeling and simulating robotic systems. It has proven easy enough to understand and use in introductory courses on robotics, kinematics, industrial automation, and mechanisms, but its advanced features also make it possible to use in complex large-scale systems.

Some areas where future improvements are planned include:

- Programming language: A future version of ROBOSIM will be embedded into a general purpose interactive programming language interpreter. This new version will also support the old command syntax for backward compatibility, but the use of a general purpose language instead of the current command interpreter will offer several advantages:

  - ✓ Program flow control statements in simulations.

  - ✓ An easier way for users to specify inverse kinematic routines for their manipulator models.

  - ✓ An easier way for users to specify arm configuration selection and collision avoidance heuristics for their models.

- Development of a graphical user interface (GUI) to control simulation options: There are some system parameters in the simulation environment which are especially suited for control by GUI methods (while keeping the current interactive commands to control them as well). These include the camera setup, lighting model, and other similar options.

- Interface to common CAD packages for importing shape designs to be used in simulations.

- Continued improvements on the path planning and arm configuration management heuristics.

## 8.0 REFERENCES

[1] Fernandez, K. R., *Robotic Simulation and a Method for Jacobian Control of a Redundant Mechanism with Imbedded Constraints*, Ph.D. Dissertation, Vanderbilt University, 1988.

[2] Springfield, J. F., *ROBOSIM Workstation Extensions*, Master's Thesis, Vanderbilt University, Spring 1989.

[3] Wilson, S. L., *Interfacing of a Robot Simulation Program with Graphic Utilities of an Intergraph Interpro 360 System*, Master's Thesis, Vanderbilt University, August 1990.

[4] Mirolo, C. and Pagello, E., "A Solid Modeling System for Robot Action Planning", *IEEE Computer Graphics and Applications*, January 1989, pp 55-69.

[5] Unimation, Inc., *User's Guide to VAL II*, Danbury, CT, 1986.

[6] Springfield, J. F., Cook, G. E., Andersen, K., and Fernandez, K. R., "ROBOSIM: A Simulation Package for Robots", *University Programs in Computer-Aided Engineering, Design, and Manufacturing*, Eds: K. P. Chong, B. R. Dewey, and K. M. Pell, American Society of Civil Engineers, 1989, pp 239-246.

[7] Biegl, C., Cook, G. E., Fernandez, K. R., and Smith, M. K., "ROBOSIM: An Intelligent Robotics Simulator", *University Programs in Computer-Aided Engineering, Design, and Manufacturing*, Ed: D. Stone, Tennessee Technological University, 1992, pp 209-216.

[8] Biegl, C., et. al, "Adaptive Control of a Dual-Arm Robot Manipulator Using On-Line Graphical Simulation", *Proceedings of ROBEXS '89, The Fourth Annual Workshop on Robotics and Expert Systems*, Palo Alto, CA, pp 253-263, 1989.

[9] Fernandez, K. R., "The Use of Computer Graphic Simulation in the Development of Robotic Systems", *Acta Astronautica*, Vol. 17, No. 1, Pergamon Press, January 1988, pp 115-122.

[10] Biegl, C., et. al, "Simulation-Based Intelligent Robotics Agent for Space Station Freedom", *Proc. of the 5th Conference on Artificial Intelligence for Space Applications*, Huntsville, AL, pp 203-210, 1990.

[11] Parlaktuna, O., Cook, G. E., Strauss, A. M., and Fernandez, K., "Jacobian control for space manipulator", *Robotics and Autonomous Systems*, No. 11, Elsevier, pp 35-44, 1993.

[12] Parlaktuna, O., Cook, G. E., Strauss, A. M., and Fernandez, K. R., "Fine Attitude Control of Space Vehicles using Space Manipulators", *International Journal of Robotics & Automation*, Vol. 9, No. 1, IASTED, pp 29-35, 1994.

# APPENDIX A: COLLISION DETECTION

In this appendix the collision detection algorithm as implemented in ROBOSIM is presented. The material presented does not include a detailed discussion of the theory behind the methods used, nor does it give an overview of collision detection. The collision detection algorithm implemented is very similar to the POCODA (POlygonal COllision Detection Algorithm) [1]. The implementation used in ROBOSIM is given with special emphasis on those extensions to POCODA.

Collision detection is very important in simulation of robots. One usually wants to know if the robot has collided with its environment or with itself. Also, as discussed elsewhere in this report, with collision detection powerful heuristics may be implemented for automatic collision free path generation.

The algorithm used in ROBOSIM can be broken down into several subalgorithms. These will be discussed from the lowest level to the highest level. The assumption used here is that all objects are defined by convex planar polygons. The problems involved in collision detection are as follows:

1. Given a polygon and a point in the plane of the polygon determine whether that point is inside of the polygon.

2. Given a polygon and a line segment determine whether the line segment crosses the plane of the polygon.

3. Given two polygons determine whether they intersect.

4. Given two objects determine whether they intersect.

5. Given two bounding volumes around two objects determine whether they overlap.

Much of the discussion that follows, as well as the actual implementation, is based on describing planes by the plane normal form of the plane equation given by

$$\Phi_{\vec{N}}(\vec{P}) = \vec{N} \cdot \vec{P} + n_d$$

where $\vec{N}$ is the normal to the plane, $\vec{P}$ is a point, and $n_d$ is the distance from the plane to the origin.

The plane described by this equation is the set of points $\vec{P}$ such that $\Phi_{\vec{N}}(\vec{P})$ is zero. Given $\vec{N}$, $n_d$, and a point $\vec{P}$, the residue $\Phi_{\vec{N}}(\vec{P})$ is zero if $\vec{P}$ is in the plane, positive if $\vec{P}$ is above the plane, and negative if $\vec{P}$ is below the plane.

The point-in-polygon problem is the most time-consuming operation. The method used to solve this problem is why the polygons must be convex. The algorithm is to follow the polygon's edges around the polygon checking to see which side of each edge the point is on. If the point is to the same side of each edge then that point is inside of the polygon. This is checked by plugging the point into each edge's penalty function. The penalty function is a plane equation such that the edge lies in the plane and the plane is perpendicular to the plane of the polygon. The penalty function is calculated once for each edge and stored in the internal structure. Given the normal to the polygon ($\vec{N}$), the directed edge ($\vec{E}$) from the first edge point ($\vec{P}_1$) to the second edge point ($\vec{P}_2$), and a point $\vec{P}$ to be tested, the penalty function for an edge is given by

$$\Psi(\vec{P}) = \vec{M}_j \cdot \vec{P} + m_d$$

$$\vec{M} = \frac{\vec{N} \otimes \vec{E}}{\|\vec{E}\|}$$

$$\vec{E} = \vec{P}_2 - \vec{P}_1$$

$$m_d = -\vec{M} \cdot \vec{P}_1$$

$\vec{M}$ is the normal vector to the penalty plane; it is the cross product of the normal to the polygon plane and the directed edge normalized with respect to the directed edge. $m_d$ is the distance of the penalty plane form the origin. This penalty function can now be used to determine which side of an edge a point is on.

The algorithm for determining if a line segment crosses the plane of a polygon should be obvious from the above discussion. The two endpoints of the line segment are both substituted into the equation of the plane in which the polygon lies. If the residues of the two points are the same sign then both points lie on one side of the plane. Therefore, the line segment did not cross the plane. If, however, the residues have different signs, then the point at which the line segment crosses the plane must be determined so as to use it in the point-in-polygon algorithm. Given two points $\vec{P}_1$ and $\vec{P}_2$ which are the endpoints of a line segment and $\Phi(\vec{P}_1)$ and $\Phi(\vec{P}_2)$ which are the residues of $\vec{P}_1$ and $\vec{P}_2$ in the polygon plane, then the point along the line segment that intersects the polygon plane is $\vec{P}_c$ where,

$$\vec{P}_c = \vec{P}_1 + \frac{(\vec{P}_2 - \vec{P}_1)\Phi(\vec{P}_1)}{\Phi(\vec{P}_1) - \Phi(\vec{P}_2)}$$

Each object in a simulation is composed of polygons, but due to speed and efficiency requirements the above tests would be prohibitively slow. Therefore, some simpler tests are required which can quickly eliminate some objects from the more exhaustive tests. The method used is to perform test on bounding boxes of the objects. A bounding box is described by a point and a vector. The point is the center of the box, and the vector is the half-diagonal vector of the box (i.e., it points from the center of the box to a corner). These values are determined by first determining the maximum and minimum values along each axis. The half-diagonal vector is calculated by taking half of the difference between the maximum and minimum along each axis. For instance, along the X axis

$$C_x = \frac{X_{max} + X_{min}}{2}$$

$$D_x = \frac{X_{max} - X_{min}}{2}$$

Now, two bounding boxes overlap if the distances between the centers along every axis is less than the sum of the half-diagonal components along the corresponding axes. However, the two bounding boxes must be defined in the same coordinate frame. Typically, each object is defined in its own coordinate frame and has a transformation matrix describing the position and orientation of the object in the world coordinate frame. Therefore, a method is needed to transform a bounding box from one frame to the other. Given two bounding boxes, $B_1 = \{\vec{C}_1, \vec{D}_1\}$ and $B_2 = \{\vec{C}_2, \vec{D}_2\}$, and two transformations $T_1$ and $T_2$ which are 4x4 matrices describing the position and orientation of boxes $B_1$ and $B_2$, respectively, let $\vec{C}_{1,2}$ and $\vec{D}_{1,2}$ be the center and half-diagonal vector of $B_1$ in coordinate frame 2.

$$\vec{C}_{1,2} = \vec{C}_1[T_1][T_2]^{-1}$$

$$\vec{D}_{1,2} = \vec{D}_1 \oplus \left([T_1][T_2]^{-1}\right)$$

where $\oplus$ is the dilation product, an operation between two matrices which can be expressed as the product of two matrices whose elements have all been changed to their absolute values.

$$\left(A_{m\times n} \oplus B_{n\times q}\right)_{ij} = \sum_{k=1}^{n} \left| a_{ik} b_{kj} \right|$$

In order to test for bounding box overlap given two boxes, one first has to express $B_1$ in coordinate frame 2 and check for an overlap. Then convert $B_2$ to coordinate frame 1 and check for an overlap. Only if both checks indicate an overlap is there one. If an overlap is indicated then further checks have to be made to determine if there is a collision.

Once a possible collision is indicated by overlap of bounding boxes, more exhaustive tests have to be performed. First, all points in one object must be transformed to the other object's coordinate frame. Once this is done, a first approach would be to check every edge in each object against every polygon in the other object. However, there are some ways to reduce the number of edges which must be checked. First, each edge in object 1 is checked against the bounding box of object 2. Only if the edge falls within the bounding box could it intersect the object. Each edge that could intersect a polygon is saved in the reduced edge array. Now, each edge in the reduced edge array is checked against the polygons in object 2. However, each polygon from object 2 is first checked to see if the plane it lies in could intersect the bounding box of object 1. If it does not, there is no need to check edges against it. Finally, each possible edge is checked against each possible polygon, using the methods outlined above, to determine if a collision exists. If not, then all points of object 2 are transformed to the frame of object 1 and the procedure repeated.

The use of this algorithm requires some special considerations when used with robots. The technique used employs a bounding box around each object in the environment, a bounding box around each link of each robot, and a bounding box around each robot. The bounding boxes around each object and each link are relatively static, but the bounding boxes around robots must be computed dynamically. This is because the bounding boxes around robots change as the joint angles in the robots change. Whenever a collision is checked for, bounding boxes are created around the robots. They are calculated by using the bounding boxes of the links. The minimum and maximum extents along the X, Y, and Z axes of the bounding boxes around the links are computed. Then a bounding box around all of these bounding boxes is computed from the minimum and maximum extents.

The purpose of bounding boxes around robots is that if there is more than one robot, even bounding box checks become expensive. If there are two robots, each with nine links (six movable and three fixed), 81 bounding box checks would be required every time. And if there were three robots, 729 bounding box checks would be required. With three robots, and therefore three bounding boxes, only three bounding box checks are required. If there is a collision between bounding boxes, only the two robots need be checked.

Another problem that requires special treatment is a collision involving the robot with itself. This is especially difficult when one considers that the design of the robot may include overlap of adjacent links. If this is the case, then if links of the robot are checked with other links of the same robot, then collisions might be seen that aren't really valid. Therefore, collisions are not checked for against adjacent links. The simulator has internal provisions for joint constraints. Therefore, any possible collision could be provided for by limiting the joint angles. However, given legal joint values, it is possible for non-adjacent links to collide. Therefore, collision detection of the robot with itself must be made. Given a nine link robot, 28 bounding box checks must be made to ensure no collisions with itself. However, this self-collision check may be controlled separately, since the user may not require these tests.

The collision detection algorithm has only two weak points. It does not handle concave polygons, and it will not signal a collision if one object is completely inside of another. The stipulation concerning concave polygons is not serious as the modeler does not generate concave polygons. The problem of not detecting a collision if one object is completely inside another derives from the fact that the algorithm used is a polygonal collision detection algorithm and not a solid model one. However, assuming two objects start off outside of each other and movements are sufficiently small, then this should not prove to be a problem. This condition also prevents the ability of one object to pass through another (i.e., a movement is large enough that two objects do not overlap at any point). The algorithm does not detect collisions in the volume swept by an object moving between positions with another object, but rather only overlap of the objects at discrete points along a trajectory.

**References:**

[1] Walter, S. E., *Polygonal Collision Detection Algorithm*, Ph.D. Dissertation, Cornell University, 1985.

# APPENDIX B:

# ROBOSIM USERS' MANUAL

## INTRODUCTION:

ROBOSIM is used for three-dimensional geometrical modeling of robot manipulators and various objects in their workspace, and for the simulation of action sequences performed by the manipulators. Models of robot manipulators, positioners, and other mechanisms with moving joints, and objects that make up the complete simulated environment are constructed in ROBOSIM's *modeling environment*. ROBOSIM also provides a *simulation environment* where every command entered by the user is executed and the results are displayed on a graphics screen. From this interactive environment users can change the simulation scenario and operate the robot manipulator models previously constructed in the modeling environment.

The ROBOSIM modeler and the ROBOSIM simulator each have a distinct language that is described in this manual. Some of the interactive commands of the simulation environment duplicate features of the modeler. For example, a box with dimensions x = 10, y = 20, and z = 30 would be constructed in the modeler with the command:

**box x=10 y=20 z=30,**

while from the simulation environment a box can be interactively constructed with the command:

**make-object box1 box 10 20 30,**

where box1 is the name given to the box created. As will be seen, there is a purpose for both of these commands. In the modeling environment the box command might be joined with other commands that generate geometric primitives to construct the links of a manipulator arm. While in the simulation environment, the box command might be used to interactively add a box to the simulated environment immediately upon execution of the command. Then the box could be moved or grasped in the same manner as a box constructed in the modeling environment. Robot manipulators or other mechanisms with moving joints must be constructed with the ROBOSIM modeler. All objects (whether constructed with the modeler or the simulator) and manipulators must be viewed with the ROBOSIM simulator.

## GETTING STARTED:

As stated above, the ROBOSIM simulator is an interactive program, the commands entered by the user are immediately executed and their results are printed or

displayed on the screen. The simulator's code is contained in one executable file named ROBO386.EXE. Typing 'robo386' at the shell prompt will invoke it. Upon startup the simulator initializes the graphics display and loads in some initial commands from the file named SETUP.CMD if this file exists in the current directory (otherwise the screen will be blank with default viewing commands). After this the simulator's prompt appears on the screen indicating that it is ready to accept commands from the user. The simulator's input and output streams can be redirected to pipes, this way it is possible to issue commands by other higher-level control programs. Currently implementations of the simulator are available for the HP 9000/300 and 9000/800 graphics workstation families, Silicon Graphics workstations, Intergraph workstations, and 386 or 486-based (386-s must have a co-processor) PC compatibles with EGA or VGA displays.

## THE ROBOSIM SIMULATION ENVIRONMENT:

The ROBOSIM package provides an interactive simulation environment where every command entered by the user is immediately executed and the results are displayed on a graphics screen. From this interactive environment users can change the simulation scenario and operate the robot manipulator models in the system. The commands available can be grouped as follows:

- *Environment configuration:* Besides the modeling services discussed above additional commands are available for the setting of global parameters like camera position, display mode, light sources, etc. The graphics display module of the simulator supports different display options like wireframe, hidden line, solid filled and shaded graphics depending on the capabilities of the hardware platform.

- *Manipulator control:* There are commands available for moving the models of manipulator arms in various modes: joint interpolated, straight line, rotation about an arbitrary axis, etc. Manipulator coordinates can be specified both in joint and world coordinates. The simulator has a built-in iterative inverse kinematics algorithm, but the user can also specify an explicit inverse kinematics method for his or her manipulator if such a method is available. Additionally, the objects in the workspace can be grasped, moved and released by the robots. If the scenario contains several manipulators these can be operated in parallel.

- *Status reporting:* Reports about different aspects of the simulator's operation (arm position, collision situations, etc.) can be obtained by using one of the appropriate commands from this group.

The command language of the ROBOSIM simulation environment has been designed with two goals in mind: (1) to provide an interactive user interface, and (2) to be usable as the interface to a higher-level task planner program. In the second application

the task planner and the robot simulator are typically interfaced using some kind of pipe mechanism and the task planner outputs similar command sequences as entered by users in interactive applications. For this reason the command language has intentionally been kept simple. A detailed description of each of the ROBOSIM simulation environment commands is given below.

## Command format:

The simulator uses a character stream command protocol. Commands can be entered from the system console, loaded from a file, or sent by another program using the pipe mechanism provided by the operating system. The general command format is:

**[<label>:] <command> <arg1> ... <argN> [; comment]**

with the following rules:

(1) One command per line.

(2) command parameters are separated by at least one white space (SP, TAB) character.

(3) There may be an optional semicolon at the end of the command, anything between this character and the end of the line is considered as a comment. Lines beginning with a semicolon are considered comment lines.

(4) There is a way to create multiple line commands, by inserting a backslash (\) character immediately before the line terminator character.

(5) There is an optional label field in the command line. Labels are immediately followed by a colon character.

(6) Movement commands can be issued only to the agents (robots) in the workspace.

(7) Multiple agent movement commands per command line are possible. The individual commands are separated by commas. Each of the commands be directed to a different agent (manipulator) in the system, which will execute the commands in parallel. The execution of the next command line begins, when the last agent finished its operation. In contrast, if commands to two different agents are placed into consecutive command lines, the execution of these commands will be sequential.

## Error reporting:

While processing a command stream, the simulator generates an error log stream. For each command line which could not be completed successfully, there will be an entry in this stream. The format of this entry is the following:

***** Error in line [label:NNN|NNN] --- <code> <message>**

If there was a label preceding the command containing the error, then the error message will contain the name of the last seen label AND the number of lines read since the last label was encountered. If the command stream did not contain labels, then the error message will contain the number of the command line counted from the beginning of the stream. (Line numbering is zero-based, that is the first line of a stream, or the line containing a label has offset 0.) Currently the following error messages are defined:

Code: Error message text:

1 Undefined command: <command>
2 Unknown object: <command>
3 Too many arguments
4 Missing argument(s)
5 Illegal argument: <index of command argument>
6 Internal error
(This error message is given if the command interpreter itself did not find any error in the command, but the execution of the command failed in ROBOSIM or in the simulation environment.)
7 Object not an agent: <command>
8 Multiple commands for the same agent: <2nd. command>
9 Illegal coordinates: <command>
(This message will be given if the desired coordinates are out of the agent's workspace, so no inverse kinematics solution exists.)
10 Joint violation: <agent> <joint index>
11 Collision: <type> <obj1> <obj2> [<jnt1> [<jnt2>]]
Where type is one of the following:
0 object to object
1 agent to object (1 joint field)
2 agent to agent (2 joint fields)
The optional joint fields are reported if agents are involved in the collision.
12 Internal movement error
(This message is reserved for unforeseen execution errors within the Simulation Library.)
13 Hand is already holding object: <object>
14 Hand is not holding object: <object>

Note that in case the error message contains the original command line, only the offending command will be included from multiple agent commands.

## Object creation commands:

**make-object <object name> <object type> <parameters>**

This command creates an object in the workspace of the simulator. Various object types have been defined, they and their parameters are described below.

**box <xsize> <ysize> <zsize> [<color>]**
**cylinder <radius> <height> [<color>]**
**cone <radius> <height> [<color>]**
**truncated-cone <radius1> <radius2> <height> [<color>]**
**sphere <radius> [<color>]**

All solid objects are created with their center of mass at the origin of the coordinate system and their principal axis parallel with the Z axis. The color parameter is used in a manner identical to its usage at the line objects.

## Object transformation commands:

**translate-object <object> x=<xtran> y=<ytran> z=<ztran>**
**rotate-object <object> x=<xrot> y=<yrot> z=<zrot>**

These commands have a slightly different argument structure which serves the purpose of using defaults. If any coordinate direction is missing from the arguments, it is supposed to be 0. The order of the arguments is up to the user, e.g.:

translate-object box1 x=12 y=23 z=5
translate-object box1 y=23 z=5 x=12

are both accepted. Transformations are performed in the order of the arguments in the argument list. (This may make a difference in the case of rotations.) The rotate command expects its arguments in degrees.

## Composite objects:

**make-composite-object <name> <object1> <object2> .....**
**link-objects <name> <object1> <object2> .....**

These commands create a new composite object by joining the objects in the argument list permanently. They do not perform any transformations on the argument objects, but simply use their current positions. The first command will create a new object

and leave the components in the workspace, while the second one will remove all components from the workspace after creating the composite object. Composite objects can be used (transformed, operated on by agents, etc..) in a manner identical to the elementary objects.

**Objects created with ROBOSIM modeling language:**

**make-robosim-object <name> <filename> <objectname> [<color>]**

This command can be used to create a composite object using its ROBOSIM source code written in the modeling language (See section **THE ROBOSIM MODELING ENVIRONMENT**). The specified file is scanned until a 'store-file' command is encountered with the specified object name as its argument. (See section on **THE ROBOSIM MODELING ENVIRONMENT** for more details.) An optional color parameter is also accepted. It will be used to color the whole object. (I.e. regardless of the complexity of the ROBOSIM model, the whole object will be colored with the same color.)

**Agents:**

**make-agent <name> <agent-type> [<color>]**

Creates an agent. Agents are basically robots whose models have been pre-compiled using ROBOSIM. Currently only one agent, the PUMA 560 manipulator, is available with inverse kinematics. The user may create agents of their own design. The PUMA 560 agent model uses the real dimensions of the arm in millimeters, so size the other objects accordingly!

**Agent positioning:**

**drive <agent> <joint angles>|<joint angle vector name>**
**move-straight <agent> <coordinates>**
**move-inter <agent> <coordinates>**
**find-path <agent> <coordinates>**
**move-straight-to <agent> <object> <coordinates>**
**move-inter-to <agent> <object> <coordinates>**
**find-path-to <agent> <object> <coordinates>**

Each of the above commands positions the agent. The coordinate specifications are agent specific, for the PUMA arm they must contain six values of either joint angles (only for the drive command) or rectangular coordinates (x, y ,z, roll, pitch, yaw) either workspace absolute or object relative. The movement can be straight line or joint-

interpolated, or the agent can be instructed to find a path to the desired location based on its knowledge of the workspace configuration. The drive command also accepts a previously recorded joint angle vector. (See later.)

**translate-agent <agent> x=<xtran> y=<ytran> z=<ztran>**
**rotate-agent <agent> x=<xrot> y=<yrot> z=<zrot>**

These commands are useful when small incremental motions of the robot arm are needed. They always perform straight-line motion. The coordinate specification uses the same scheme as the object transformation commands, that is the coordinates are named, and any unspecified coordinate direction is assumed to be 0.

**minimal-step <value>**

This command sets the robot movement simulation step size for the movement commands. The value specified determines the speed at which the robot(s) will move. A low value, e.g., 1, will result in slow robot motion, while a higher value, e.g., 10, will give fast motion.

**set-solution <agent> <value>**

This command selects the inverse kinematics solution used for the agent. The current inverse kinematics method for the PUMA 560 arm provides 8 different solutions for (almost) any location. Some of these solutions are typically invalid due to joint angle constraints. The accepted range for 'value' is 0...7. The argument to the 'set-solution' command sets the configuration in the following way:

| | | |
|---|---|---|
| right handed: | 0..3 | (bit 2 = 0) |
| left handed: | 4..7 | (bit 2 = 1) |
| | | |
| elbow down: | 0,1,4,5 | (bit 1 = 0) |
| elbow up: | 2,3,6,7 | (bit 1 = 1) |
| | | |
| wrist down: | 0,2,4,6 | (bit 0 = 0) |
| wrist up: | 1,3,5,7 | (bit 0 = 1) |

The command also accepts the special value of -1 which instructs the agent to select the most suitable solution automatically. This is the default operation of the agent. Note that setting a fixed configuration index will more likely result in joint limit violation error messages, since the agent has no chance for switching solutions. For small movements the automatic selection is based upon choosing a valid configuration which is closest to the current joint variable values. This strategy works best when the agent is performing various tasks in a relatively small part of the workspace. However for a major position shift this may not be the best approach. In such cases the agent will select the new configuration which offers the most room (i.e., all joint angles are as far from their

respective limits as possible) for moving around in the vicinity of the new location. To determine which strategy to use the agent compares the joint angles of the old and new positions. If the difference is larger than a preset threshold, the second method is used, otherwise the first. This threshold value can be set with the following command (default value is 45 degrees):

**set-large-move-limit <limit>**

In most of the cases this strategy should work fine. However it is possible that in some cases explicit control of the robot arm configuration is necessary. (It is most likely to occur if relatively large straight-line motion segments are needed. In straight-line motion mode the simulator considers a configuration change an error, since it would result in an abrupt reorganization of the links during the motion segment.) For such situations the simulator offers the following configuration management commands:

**get-solution <agent>**

Prints out the currently used configuration index. Note that you will get a value between 0 and 7 even if you use the automatic selection method.

**get-valid-solutions <agent>**

Prints the indices of all valid configurations for the current positions.

**freeze-solution <agent>**

Is equivalent to using 'set-solution' with the value obtained by using 'get-solution'.

The simulator also performs collision testing while moving the robots in the workspace. The collision testing can be enabled/disabled with the following command:

**set-collision-check <flag>**

If the flag value is non-zero, collision checking is turned on, otherwise it is turned off. Initially the collision checking is enabled.

**Position reporting:**

**get-position <agent>**
**get-angles <agent>**
**record-angles <agent> <joint vector name>**

These commands write a line of the following format to the report stream:

**position of agent <agent> in line [label:NNN|NNN] --- <pos>**

These commands behave differently based on the operating mode of the simulator. If the simulator is in simulation only mode, then the manipulator model's joint angles or coordinates are reported. If the simulator is connected with a real robot manipulator then the robot hardware is queried for the actual joint angles, the simulator's model is updated with the reported angles, and these angles are printed out. This way the usage of these commands will synchronize the simulator's model with the actual manipulator. For the line numbering convention in the report stream see the explanation at the error report messages. The get-position command prints the position in world (rectangular) coordinates, while the get-angles command in joint angles (in degrees). The record-angles command is similar to the get-angles command, but it also records the angles in a coordinate vector which will be associated with the symbol specified in the command line. Coordinate vectors can be used as parameters for drive commands, and they are generally useful for recording important locations in the robot's workspace. The coordinate vector data base can be saved and restored with the following two commands:

**save-positions <filename>**
**load-positions <filename>**

As an additional safety measure against inadvertently losing important data, the simulator automatically saves the current coordinate vector set into the last used file if either a save or a load command was executed previously.

## Grasping:

The simulator's grasping operations are based on grasping attributes associated with each object in the workspace. These are the grasping coordinates -- in an object relative coordinate frame -- and the grasping opening used to establish contact with the object. The grasping attributes are best specified immediately after object creation, when its location is still known, and then the grasping coordinates will be transformed any time the object is moved. Commands:

**define-grasping-point <object> <x> <y> <z> <roll> <pitch> <yaw>**

Establishes the grasping point and hand orientation in an object-relative coordinate frame.

**define-grasping-opening <object> <distance>**

Establishes the hand opening which is used to grasp the object. The hand is CLOSED to the above distance upon grasping...

**define-default-grasping-gap <distance>**

This is the EXTRA hand opening above the value specified above when the hand is moving in to grasp the object. This is a global value, but may be overwritten for individual object by using...

**define-approach-opening <object> <distance>**

The default approach opening (= grasping opening + default gap) can be overwritten for individual objects using this call.

**move-to-grasp <agent> <object>**

This command moves the hand to the grasping point of the object and opens it to the approach opening (defined using either the default gap or the individual approach opening commands). It will give an error message if the hand already holds an object. This command uses the 'find-path' command's method to get to the desired point.

**grasp <agent> <object>**

This command grasps the selected object. The hand is already supposed to have been moved to the grasping point of the object, if not, an error message is generated. If the hand is not empty an error message is generated. Otherwise the hand closes to the grasping opening associated with the object, and in the simulation's data base a temporary link is set up between the object and the last link of the robot manipulator.

**release <agent> <object>**

This is the opposite of grasp. It gives an error message if the hand is not holding the specified object. NOTE: the simulator does not model effects like gravity, etc.. If an object is released in the 'air' it will stay there in the simulated environment, but of course it will drop in the real world --leading to inconsistencies between the world and its model.

**General graphics setup:**

**look-from <x> <y> <z>**
**look-at <x> <y> <z>**
**twist-camera <angle> <incremental>**

Establish parameters for the viewing transformation. The twist-camera command is used only in the SGI implementation, where it rotates the camera around its axis. If the second parameter is given (its value is not important, it just serves as a place holder) then the twisting is done incrementally to the current camera angle, otherwise the angle parameter is interpreted as an absolute angle. In the HP-UX version this command has no effect.

**define-color <name> <r> <g> <b>**

Associate a color specification with 'name'. Red green and blue intensities range from 0.0 to 1.0.

**light-source <x> <y> <z> <color> <ambient>**

Specify light source. They are not needed for wire frame display, only for the other types. The ambient parameter just serves as a place holder, its value is not important. If it is present then the light source is ambient, otherwise it is directional.

**display-type wireframe|hidden|solid|shade**

Define graphics display option to be used. Not all implementations support all display modes, if an unsupported mode is selected, the command is silently ignored.

**Physical manipulator control:**

**enable-execute <agent1> <agent2>**

Enable sending commands to the real robots in the system. The execution will be enable for those manipulators only which are listed in this command. Their order must match the order in which the low-level interface function expects the coordinate vectors.

**disable-execute**

Opposite of the above -- only simulated execution afterwards. The system starts up in this state.

Currently the simulator is only capable of controlling the PUMA 560 manipulator with the UNIMATE controller. Parts of the robot control interface run on an IBM PC AT (or compatible), to satisfy certain real-time requirements. Before the 'enable execute' command is invoked, the robot control program running on the IBM PC must be started. Its name is typically 'ROBOC'. After startup, the user does not have to do anything with it. This program can also be used as a simple terminal emulator for the UNIMATE controller. The IBM PC interface program has a self-explanatory help menu (which can be displayed by typing the Alt-H key combination), so a more detailed description of its functions is not included here. NOTE: for the current SGI implementation of the simulator the serial interface connecting the IBM PC and the graphics workstation must be initialized. (Due to a bug in the serial driver on these machines.) This is done by entering the robot control program's terminal emulator mode for the graphics workstation (see help menu), logging in, and entering the 'sleep' command with a long timeout value.

## General:

**load <filename>**

Take commands from the specified file. Returns when end of file or the 'exit' (or 'end') command is encountered. Loads may be nested.

**set-echo <flag>**

If 'flag' is non-zero then all subsequent loads will echo the contents of the command file as it is being processed.

**exit** or **end**

If given at the simulator prompt, exits the simulator program returning to the DOS prompt. If given in a command file, returns to the simulator prompt.

**abort**

If given at the simulator prompt, exits the simulator program returning to the DOS prompt in the same manner as the **exit** or **end** command. If given in a command file, exits the simulator program returning to the DOS prompt.

## THE ROBOSIM MODELING ENVIRONMENT:

Similar to other solid modeling software tools, ROBOSIM models the three dimensional geometric objects using lists of their bounding polygons. The ROBOSIM modeling language is used to specify complex geometric shapes which are used as manipulator links or as passive cylinders, cones, extruded polygons, etc. Translational, rotational and scaling transformations are used to combine these objects to form the desired shape.

The ROBOSIM modeling language mimics a "geometric microprocessor" which has a few graphics registers ("a" through "d") containing the three dimensional polygon lists of the complex object being built. There is a designated accumulator register ("a"). Whenever, a new elementary shape (box, etc.) is created, it is concatenated to the contents of the accumulator register. Similarly to the basic shape generation primitives, the geometrical transformations also operate on the contents of the accumulator register. This, together with the ability to move or concatenate geometric data to or from the other registers enables the definition of arbitrary complex shapes in terms of a few basic operations. Link coordinate frames can also be added to the contents of the accumulator register thus making it possible to specify the geometric transformations associated with the links of a manipulator. The ROBOSIM modeling language is described below.

## Solid primitive commands:

All solid objects are created with their center of mass at the origin of the world coordinate system and their principal axis parallel with the z-axis. In each of the following commands, the order of the arguments is up to the user.

**box x=X y=Y z=Z**

A rectangular parallelepiped with dimensions as specified is generated in the "a" register with appropriate mass properties.

**cylinder r=R h=H**

A cylinder with circular faces perpendicular to the z-axis with the specified dimensions is generated in the "a" register.

**cone r=R h=H**

A cone with circular face in the xy-plane with the dimensions specified is generated in the "a" register.

**truncated-cone rl=RL rh=RH h=H**

A truncated cone with lower circular face in the xy-plane with the dimensions specified is generated in the "a" register.

**sphere r=R**

A sphere of radius, R, is generated at the origin of the "a" graphics register.

## Graphic register control commands:

**clear**

Clears the contents of the "a" graphics register.

**store b or c or d**

The contents of the "a" graphics register are stored in one of the temporary graphics registers (b,c,d). The contents of the "a" register are unchanged.

**load b or c or d**

The contents of the b or c or d register are copied into the "a" graphics register destroying its previous contents and leaving the b or c or d register unchanged.

**exchange b or c or d**

The contents of the b or c or d register are exchanged with the contents of the "a" graphics register.

**add b or c or d**

The contents of the b or c or d register are appended to the contents of the "a" graphics register leaving the b or c or d register unchanged.

**Surface generation commands:**

**vector x=X y=Y z=Z**

Generates a vector from the origin to (X,Y,Z) in the "a" graphics register.

**move x=X y=Y z=Z**

Generates non-visible vector from previous vector tip to (X,Y,Z) in the "a" graphics register.

**draw x=X y=Y z=Z**

Generates a visible vector from the previous vector tip to (X,Y,Z) in the "a" graphics register.

NOTES:

- In the above commands the order of the arguments is up to the user, e.g.:

  vector x=100 y=50 z=200
  and
  vector z=200 x=100 y=50

  are both accepted.

- For the current version of ROBOSIM, a minimum of two connecting (end-to-end) lines must be drawn, and they cannot be collinear (an exception to this constraint is a single line followed by the REV-SURFACE command). This is due to the built-in collision detection algorithm that checks normals to polygon

surfaces (a minimum of two connecting lines is required to define a surface). Future versions may be modified to permit the user to exclude certain objects from the polygon-normal check.

**rev-surface**

Generates a surface by revolving the contents of the "a" register about the z-axis. A line will become a surface or solid, a surface will become a solid. Prototype curves should be defined in a plane containing the z-axis to insure that the mass-properties computations are correctly computed.

**extrude-surface z=Z**

Generates a surface by sweeping the graphical contents of the "a" register along the z-axis. A line becomes a surface, a polygon becomes a solid. The extrusion will be from -Z/2 to +Z/2.

**Model manipulation commands:**

**translate x=X y=Y z=Z**

Translates the graphical contents of the "a" register the specified relative distance.

**rotate x=X or y=Y or z=Z**

Rotates the graphical contents of the "a" register by the specified angle in degrees about the x,y, and z axes of the world coordinate frame.

**scale x=X y=Y z=Z**

Multiplies the graphical contents of the "a" register by the specified amounts along the x,y, and z axes.

In each of the above model manipulation commands, the order of the arguments is up to the user.

**File control commands:**

**execute-file fname**

The ROBOSIM commands contained in the external file, fname, are placed in place of this instruction. All commands are allowed except "execute-file" itself.

**store-file fname**

Stores the graphical and mass properties contained in the "a" register into the file specified. The contents of the "a" register are unchanged.

**store-link robot-name.([LOC][L0][L1]...)**

This command adjusts the link data contained in the A-register such that the coordinates are consistent with Hartenburg-Denavit link definitions. The data is then stored in file robot-name.EXT, where EXT<(LOC,L0,L1,L2,L3...LN). LOC indicates the specification for the robot's base location, while L0...LN contain the data for the first N links of the robot.

## Special commands:

**end**

Terminates execution of the ROBOSIM process and returns to the monitor.

**set-nface N=n**

Sets the number of facets generated to approximate a circular surface. This specification affects the solid primitive routines and the rev-surface command.

**set-density d=D**

Sets the default mass-density for solids defined by the primitives, extrude and rev-surface command. A negative value may be used to create a "hole" in a previously defined solid or a zero may be specified to create a massless graphic entity.

## Link joint specification:

**(r, p or f)-joint-(i or i+1)**

A revolute, prismatic, or fixed joint is specified by the prefix r, p, or f, respectively. The suffix, "i", indicates the joint is closer to the robot base than the joint specified with suffix, "i+1". As an example, a revolute joint defined in a robot link located on the base side of link would be specified by the command, r-JOINT-i.

# APPENDIX C:

# ROBOSIM TUTORIAL

## Getting Started:

Before attempting to operate ROBOSIM, you should print a copy of the ROBOSIM manual. Two formats are provided: MANUAL.WP and MANUAL. MANUAL.WP is formatted in WordPerfect 5.1; MANUAL is formatted in plain DOS text. The ROBOSIM manual contains a description of all of the commands available in the interactive simulator mode, and a description of all of the commands used in constructing models with the ROBOSIM modeling language.

To help you get started, let's first examine the SETUP.CMD file in the ROBOSIM root directory. You may wish to obtain a listing of this file by printing a copy. Lines beginning with a semicolon are considered comment lines. You will note that the first set of non-comment lines are color definitions. Other colors could be defined by altering the weights of the red green and blue fields. The next set of commands are the view commands. As you use the simulator, you will find frequent need to change the look-at and look-from commands to alter your perspective of the simulated environment. These two commands will be your way of "walking around the simulated environment" to alter your view. Following the view commands is the collision-check command. Here collision checking is turned Off by setting the argument of the set-collision-check command to 0 (any other number would turn On the collision checking algorithm). The next command (minimal-step) in the SETUP.CMD file sets the speed of operation. This commands sets the robot movement simulation step size for the movement commands, and hence, controls the speed of operation. A small step size, e.g. 1, results in a slow speed. A large step size, e.g., 10 results in a faster speed. You will need to try different values for the minimal-step size to achieve the desired speed of operation for your system. Finally, the last two commands create the "floor" of the simulation by creating a box of dimensions x=2000, y=2000, and z=60. The box created with the make-object command is centered at its center of gravity. The second command translates the floor down by 30 units so that its top is in the z=0 plane. (The x-axis is right (-) to left (+). The y-axis is in (-) and out (+) of the screen. The z-axis is bottom (-) to top (+).) The final command in the SETUP.CMD file is an end statement, which should always be present as the last statement. The SETUP.CMD file will normally be used to build the entire simulated environment (workspace, robots, positioners, etc.). A SETUP.CMD file tailored to the user's specific needs will be a part of almost all applications of the ROBOSIM simulator. Other command files will then be used to operate the machines setup in the SETUP.CMD file. While this procedure is often followed, it is not necessary, however, as we will see.

We will now load the simulator in preparation for demonstrating some of the interactive commands. To do this we should first type 'init' at your ROBOSIM directory. This only needs to be done once after you first turn your system On. This sets the driver and specifies the path where the fonts are located. These two commands may be included

in your autoexec.bat file, if you wish to eliminate the need for the initial 'init' command. To run the simulator type 'robo386'. The interactive window with robag> prompt and floor constructed in the SETUP.CMD file will be displayed on the screen. Summarizing, the sequence of commands to reach this point are:

<c:\> cd robosim
<c:\robosim> init
<c:\robosim> robo386

As we go through the remainder of this tutorial, we will introduce changes to the SETUP.CMD files using your text editor. As these changes are introduced, they must be made at that time in order that subsequent demonstrations of commands will work. We will also be demonstrating how additional "data" files can be constructed for modeling objects, robots, positioners, etc., and how "command" files can be constructed for commanding motion sequences of the simulated cell.

**Creating and Driving a Robot:**

We can create a robot in our simulated environment by typing the following command at the robag> prompt:

robag>make-agent r1 puma560 yellow

This command will create a yellow, puma robot (named r1) with its base located at x=0, y=0, z=0. As stated in the manual, under the 'make-agent' command, currently only one agent is available, the PUMA 560 manipulator. We may build other robots or positioners but we can only move them with the 'drive' command. All of the other commands require the inverse kinematic solutions which are currently only available for the PUMA. Future versions of ROBOSIM will permit the user to incorporate inverse kinematic solutions for any robot or positioner.

To demonstrate the positioning commands available to move the robot, type the following command at the robag> prompt:

robag>drive r1 90 -90 90 0 0 0

The robot will move to a straight-up position. Remember that you can change the speed of its motion by changing the step size with the 'minimal-step' command. In the 'drive' command, the 6 values following the robot name are the joint angles you wish to drive the robot to. All six angles must be specified even if some are zero. To drive the robot back to its home position, type:

robag>drive r1 0 0 0 0 0 0

We may reduce the work involved in commanding the robot to drive to a specified set of

joint angles by using the 'record-angles' command. For example, with the robot at its home position, we may type:

```
robag>record-angles r1 h
```

This will create a joint vector named h (for home). Now if we move back to the vertical position by typing,

```
robag>drive r1 90 -90 90 0 0 0
```

we can return to the home position by simply typing,

```
robag>drive r1 h
```

We could have defined a joint vector for the vertical position of the robot as well. Indeed, we can define as many joint vectors (corresponding to different positions of the robot) as we wish. Then we can drive to a desired position by simply specifying its joint vector name in the 'drive' command. If you wish to save your set of joint vectors, you may do so with the 'save-position' command. Then, each time you load the simulator, you can also load the joint vectors with the 'load-positions' commands. This command could also be included in the SETUP.CMD file so that the joint vectors would be automatically loaded. To demonstrate, let's drive the robot to the vertical position and define a joint vector for that position:

```
robag>drive r1 90 -90 90 0 0 0
robag>record-angles r1 v
```

We may now drive between the vertical and home positions by typing,

```
robag>drive r1 h
robag>drive r1 v
```

To save these two joint vectors type,

```
robag>save-positions pos
```

This will save the 'h' and 'v' joint vectors in a file named 'pos'. The file 'pos' will be created in your ROBOSIM directory. A listing of this file will show that it contains two lines defining the 'h' and 'v' joint vectors. These vectors may be loaded at any time from the robag> simulator prompt by typing,

```
robag>load-positions pos
```

As stated above, this command could also be included in the SETUP.CMD file, so that the 'h' and 'v' joint vectors would be automatically loaded upon entering the simulator

program.

At this point, let's exit the simulator program (by typing 'exit' or 'end' at the robag> prompt. Before reentering the simulator program, let's modify our SETUP.CMD file so that our robot will be automatically created, and our joint vector file will be automatically loaded. If you wish to save the original version of the SETUP.CMD file, you may copy it to SETUP.ORG (for original setup file). To modify the SETUP.CMD file, use an editor to add the following two lines (boldface) immediately before the 'end' command.

.
.
.
translate-object floor z=-30
**make-agent r1 puma560 yellow**
**load-positions pos**
end

Let's also turn On the collision-check algorithm by modifying (with your text editor) the 'set-collision-check' command in the SETUP.CMD file to read:

set-collision-check 1

Now when you enter the simulator program by typing robo386, you should see the robot and the floor. You can also, command the robot to drive to the vertical position by simply typing 'drive r1 v', since the joint vector 'v' has been loaded. You may likewise drive to the home position by simply typing 'drive r1 h'.

**Robot Positioning Commands:**

As you will find in reading the ROBOSIM manual, there are a number of commands used for positioning the robot (agent). We will look at some of these now. Consider the 'translate-agent' command. This command always performs straight-line motion. To demonstrate, move the robot to the home position, and then type,

robag>translate-agent r1 x=300

The robot will move in a straight line in the x-direction, maintaining the same orientation of the end-effector. We can obtain the position of the end-effector with the 'get-position' command,

robag>get-position r1

The simulator will respond with

Agent r1 coordinates: 711.48 149.10 1160.78 0 0 180

The first three numbers give the xyz position of the end-effector. The last three numbers give the orientation of the end-effector in terms of roll, pitch, and yaw.

We can also get the joint angles at this position of the robot with the 'get-angles' command,

robag>get-angles r1

The simulator response will be

Agent r1 joint angles: 0 -15.56 61.22 0 -45.66 0

Next, use the 'translate-agent' command to move in the negative x-direction with,

robag>translate-agent r1 x=-1400

Again, use the 'get-position' and 'get-angles' commands to obtain the position and joint angles at this location of the end-effector. The results are,

Agent r1 coordinates: -688.52 149.10 1160.78 0 0 180
Agent r1 joint angles: 0 -127.83 52.78 0 75.06 0

Let's now use the 'move-inter' command to move back to the left-hand position. The 'move-inter' command moves the robot with linear joint interpolation. With this command, we specify the position and orientation (3 position values and 3 values for roll, pitch, and yaw) that we wish to move to. Each joint moves linearly to accomplish this. The speed of each joint is such that all joints start and stop motion at the same time. The 'move-inter' command is,

robag>move-inter r1 711.48 149.10 1160.78 0 0 180

You will observe that the robot moves in an arch to get to the specified position. With joint-interpolated motion the trajectory of the end-effector is not specified between the start and stop positions.

We will now return to the right-hand position using the 'move-straight' command. Again, we specify the position and orientation that we wish to move to. With this command, however, the robot will move along a straight-line between the start and stop points. To return along a straight-line to the right-hand position, we issue the command,

robag>move-straight r1 -688.52 149.10 1160.78 0 0 180

Now, the end-effector moves along a straight-line to the specified position, just as it did with the 'translate-agent' command.

Next, we will examine the 'find-path' command. This command triggers a set of heuristic rules that will attempt to move the robot from its initial position to the specified final position without any collisions with objects in the workspace. We will first issue the command to find a collision-free path to the previous left-hand position of the robot,

robag>find-path r1 711.48 149.10 1160.78 0 0 180

We observe that the end-effector moves along the same arch as with the 'move-inter' command.

Now let's construct a box and position it such that the arched path results in a collision. The algorithm triggered by the 'find-path' command will then search for a new collision-free path to the specified final position. We first construct the box (named b1) and translate it to a position that will result in a collision:

robag>make-object b1 box 100 100 100 green
robag>translate-object b1 y=149.10 z=1300

Now let's issue the find-path command instructing the robot to find a path from the left-hand position (711.48 149.10 1160.78 0 0 180) to the right-hand position (-688.52 149.10 1160.78 0 0 180):

robag>find-path r1 -688.52 149.10 1160.78 0 0 180

You will observe that the robot first tries to move to the specified right-hand position by moving in an arched path but this results in a collision with the box. A collision-free, built-in path planning algorithm then takes over and tries to maneuver past the box with the arm folded. For this position of the box, this maneuver works and the robot is able to move to the final position. Then the robot returns quickly to the start position and repeats the learned collision-free path in one continuous motion. If you were to lower the box by 50 units in the z-direction (translate-object b1 z=-50), several minutes may be required (varies with the speed set by the minimal-step setting) for the robot to find a collision-free path. You may or may not want to try this because of the time required to complete the search. If you do try a lower box position, and wish to stop the motion prior to completion of the search, you may do so by pressing 'control-c'. This will leave you in the graphics screen mode, so you should then type 'robo386' and then 'exit' at the robag> prompt in order to make an orderly exit back to DOS.

Let's now exit and then reenter the simulator after modifying the SETUP.CMD file again. We will then demonstrate the commands for moving to a specified object. For the demonstration we will let the object be a box of dimensions x=300 y=100 z=400. We could construct the box interactively from the simulator robag> prompt. Instead of doing this, however, we will again modify the setup.cmd file to include the box. The additional commands to add to the SETUP.CMD file with your editor are shown below in boldface.

.
.
.
translate-object floor z=-30
make-agent r1 puma560 yellow
load-positions pos
**make-object b1 box 300 100 400 cyan**
**translate-object b1 x=-500 z=201**
end

The two additional commands will create a box (cyan color) of the desired dimensions and translate it 500 units in the negative x-direction and 201 units in the positive z-direction. The latter command will position the box 1 unit above the floor. Now when we enter the simulator program by typing 'robo386', we should see the robot located at the center of the floor and the box b1 positioned one unit above the floor and 500 units to the right of the robot.

We will now use the 'move-inter-to' command to move the robot's end-effector to a position 1 unit above the box, centered on the y-axis, and 100 units to the right of the y-axis along the x-axis. The complete command is

robag>move-inter-to r1 b1 -100 0 201 180 0 0

The first three numerical arguments of this command specify the desired position of the end-effector relative to the box's coordinate frame. The box's coordinate frame is at its center of mass. Thus the specified position (-100 0 201) will locate the end-effector 1 unit above the box and 100 units along the negative x-axis (right of center). The last three numerical arguments of the 'move-inter-to' command specify the orientation of the end effector in terms of roll, pitch, and yaw about the x, y, and z axes of the world coordinate frame. A roll of 180 degrees about the x-axis is required to avoid a joint violation. We can obtain the position and orientation of the end-effector in world coordinates by issuing the 'get-position' command,

robag>get-position r1

The simulator responds with, the message

Agent r1 coordinates: -600.00 0.00 402.00 -180.00 0.00 0.00

The robot is centered on the world coordinate frame. The center of the box was translated 500 units along the negative x-axis, and the position specified to locate the end-effector in the box coordinate frame was -100 units along the x-axis. Thus in absolute coordinates, the x-axis position of the end-effector is -(500+100) = -600 as obtained from the 'get-position' command. Likewise, we specified the z-axis position relative to the box

frame as 201, which is 1 unit above the box. Since the box was originally translated to a position 1 unit above the floor and is itself 400 units high (z-direction), the absolute position of the end-effector is 201+201 = 402 as obtained from the 'get-position' command. The roll angle is -180 even though we specified 180. The system does not distinguish between the two.

Next, let's try to reposition the end-effector so that it is positioned to the left of the box's center by 100 units and maintains the same y and z coordinates. If we try to accomplish this with the 'move-inter-to' command, a collision will be detected, and the robot will not move. This is because the 'move-inter-to' command attempts to make the commanded move in an arc that causes a collision between the robot and the box. (Remember that the end-effector is located only 1 unit above the box.)

We can avoid this problem by use of the 'move-straight-to' command,

robag>move-straight-to r1 b1 100 0 201 180 0 0

Now you will observe that the end-effector moves in a straight line, at a spacing of 1 unit above the box, to the specified location.

We could have also accomplished the last move with the 'find-path-to' command. To demonstrate, let's move back to the right-hand side of the box with the command,

robag>find-path-to r1 b1 -100 0 201 180 0 0

We observe that the robot moves up and outward to avoid a collision with the box in moving to the specified position.

We have now discussed most of the simulator commands available for positioning the robot. The remaining commands, that we will discuss next, involve grasping and moving into position to grasp. Before demonstrating these commands, let's exit the simulator by typing 'exit' at the robag> prompt and then reenter the program by typing robo386 at the DOS prompt. With the SETUP.CMD file, as previously modified, the robot should appear with the box b1 located to its right.

Let's now define a grasping point on the box and move the robot into position to grasp the box. Once we have done this, we may grasp the box and move it to another location.

The grasping point is defined with the following command,

robag>define-grasping-point b1 0 0 201 180 0 0

This defines the grasping point to be centered on the top of the box and 1 unit above the top.

We can now move the robot into position to grasp the box with the command,

robag>move-to-grasp r1 b1

The robot will be observed to move into position to grasp the box. To grasp the box, we issue the command,

robag>grasp r1 b1

The box is now attached to the end-effector and will move with the robot. To demonstrate let's return the box to the home position, release it, and swing the robot out of the way. The commands are,

robag>drive r1 h

robag>release r1 b1

robag>drive r1 135 0 0 0 0 0

The simulator knows that the box has been moved to a new location. To demonstrate let's use the 'move-inter-to' command to move the end-effector to the lower right side of the box. Remember that this command is given in terms of the object coordinate frame. The command is,

robag>move-inter-to r1 b1 100 0 201 180 0 0

The robot is observed to move its end-effector to the designated position on the box. Now let's exit the simulator by typing 'exit' at the robag> prompt.

**Using the ROBOSIM Modeler:**

So far, we have used commands available in the ROBOSIM simulator to create robots, make elementary objects, and move them about with the robots. If we wish to model a more complex object or if we wish to model a robot (other than the Puma 560 which is available) or positioner with moving joints, we must do so with the ROBOSIM modeling language. Once we have created the model, we can then view, move, and use the model in the ROBOSIM simulator. In this section we will discuss the basic steps required for creating models with the ROBOSIM modeler and calling them with the simulator.

We create models with the ROBOSIM modeler using an editor and the commands listed under the section **THE ROBOSIM MODELING ENVIRONMENT**. To demonstrate, use your editor to create a file called SOLIDOBJ.DAT with the following commands:

```
move x=200 y=0 z=0
draw x=0 y=-400 z=0
draw x=-200 y=0 z=0
draw x=200 y=0 z=0
extrude-surface z=400
store-file trisolid
end
```

The first command generates a non-visible vector from the previous vector tip (in this case, the origin) to (X,Y,Z) in the "a" graphics register. The next command then generates a visible vector from (200,0,0) to (0,-400,0). The third command generates a visible vector from (0,-400,0) to (-200,0,0), and the fourth command completes the formation of a triangular polygon with vertices (200,0,0), (0,-400,0), and (-200,0,0). The fifth command generates a solid triangular object by sweeping the triangular polygon contained in the "a" register along the z-axis. The solid generated will be 400 units long, extending from -200 to +200 along the z-axis. The sixth command stores the graphical contents of the "a" register (in this case, the triangular solid) into the file called 'trisolid'. The final 'end' command terminates the ROBOSIM file.

The suffix '.DAT' on the SOLIDOBJ.DAT file that we have created is recommended but not necessary. The recommended practice is to use the '.DAT' suffix for ROBOSIM data files such as this one, and the '.CMD' suffix for simulator command files such as in the SETUP.CMD file. We will have more to say about other types of command files in a later section.

Before we can use the object created in the SOLIDOBJ.DAT file we must call for it from the simulator program. To do this we make use of the 'make-robosim'object'. This may be done interactively from the simulator robag> prompt, or the command may be inserted into the SETUP.CMD file. Using the latter alternative, let's use an editor to modify the SETUP.CMD file as follows (new commands in boldface print),

```
.
.
.
translate-object floor z=-30
make-agent r1 puma560 yellow
load-positions pos
make-object b1 box 300 100 400 cyan
translate-object b1 x=-500 z=201
make-robosim-object obj solidobj.dat trisolid green
translate-object obj x=500 z=201
end
```

The 'make-robosim-object' command has three arguments plus an optional color

specification. The first argument is the name we wish to give the object in the simulator. Here we have called it "obj". The second argument is the file name of the ROBOSIM file we previously built with the modeling language to create the triangular solid. The third argument is the object name specified by the 'store-file' command in the SOLIDOBJ.DAT file. The 'make-robosim-object' command causes the SOLIDOBJ.DAT file to be scanned until the 'store-file' command is encountered with the specified object name (in this case, 'trisolid') as its argument.

Once we have used an editor to build the SOLIDOBJ.DAT file and have modified the SETUP.CMD file to call this file, the triangular solid will appear on the simulator display when we run the simulator by typing 'robo386' at the dos prompt. The triangular solid will appear 500 units to the left of the robot and 1 unit above the floor, as set by the 'translate-object' command above.

The triangular solid, obj, can now be grasped, moved, etc., just as elementary objects constructed directly in the simulator. As previously stated, the ROBOSIM modeler provides much greater flexibility in building complex shapes than can be done directly in the simulator, and the modeler must be used for creating parts with moving joints, such as robots and positioners.

**Using a Command File:**

So far, we have operated our robot by entering commands one by one in the simulator's interactive window. Let's now develop a command file containing a sequence of commands, and show that this can be called by the single command 'load <filename>'. Let's use our editor to build a file called SWITCH.CMD with the following commands:

```
define-grasping-point obj 0 -100 201 180 0 0
define-grasping-point b1 0 0 201 180 0 0
move-to-grasp r1 obj
grasp r1 obj
record-angles r1 ang-1
drive r1 90 -28.48 196.73 0 11.75 -28.31
record-angles r1 ang-2
release r1 obj
move-to-grasp r1 b1
grasp r1 b1
record-angles r1 ang-3
drive r1 ang-1
release r1 b1
drive r1 ang-2
grasp r1 obj
drive r1 ang-3
release r1 obj
drive r1 h
```

end

The objective of the above command file will be to use the robot to switch the places of the two objects b1 and obj. To do this we will first move obj out of the way. Then we will grasp b1 and move it to where obj was originally. Finally, we will retrieve obj from its temporary location and move it to where b1 was originally. We then drive r1 to its home position and end the program. In lines 1 and 2 we define the grasping point for objects obj and b1. In line 3 we move r1 into position to grasp the object obj, and in line 4 we grasp it. So that we will be able to return r1 to this same location, we record its angles in the file ang-1 in line 5. We then move the object obj out of the way by swinging the robot 90 degrees (first joint) with the drive command in line 6. (The other angles of the drive command were obtained by observing the angles recorded in the 'record-angles' command when done interactively from the simulator's window.) This will be the temporary location for parking obj while we move b1 into its previous location. So that we will be able to return to get it, we record r1's angles in line 7 and store them in file ang-2. We then release obj in line 8, and move to grasp b1 in line 9. In line 10 we grasp b1, and in line 11 we record r1's angles in file ang-3. Then in line 12 we drive r1 to ang-1, release b1 in line 13, and drive r1 to ang-2 to retrieve obj in line 14. In line 15 we grasp obj, and drive r1 to ang-3 in line 16. We release obj in line 17, and drive r1 to the home position h in line 18. Finally, we end the command file with an 'end' statement.

To run this command file, enter the simulator by typing 'robo386' at the DOS prompt. Then at the robag> prompt type 'load switch.cmd'. If everything is done correctly, the robot should switch the locations of the two objects and return to its home position. Type 'exit' at the robag> prompt to return to DOS.

**Examples:**

The reader is referred to the EXAMPLE files for more extensive examples of using the ROBOSIM modeler to develop complex objects and robots. Further explanations of the various modeler commands is given in these examples. The reader's attention is particularly directed to the 'ARM.DAT' file in EXAMPLE3 for a discussion of the link joint specifications, and how to use these commands in building rotary or prismatic joints. The reader may also wish to print out a listing of the PUMA560.DAT file to see how the robot was constructed and how the link joints were assigned.

# APPENDIX D:

# ROBOSIM INSTALLATION INSTRUCTIONS

## System Requirements:

This diskette contains a PC version of the graphical robot simulation program ROBOSIM. The system requirements are:

- A 386 (with 387 math coprocessor) or a 486 cpu
- About 2 Megabytes of RAM (the program will run in less memory, but then it will swap to disk)
- The system MUST NOT have any program (driver or TSR) installed which puts the 386 into protected or virtual 8086 mode. Examples of such programs are expanded memory emulators, some debugger drivers, WINDOWS in 386 mode, etc. The only exceptions to this rule are VCPI compliant EMS emulators like QEMM.

## Installation:

The following installation instructions assume that you will install ROBOSIM on your hard drive (c-drive) in a directory ROBOSIM and that you will copy the files on this diskette from your a-drive. If your drive designations are different or if you wish to install the program into a different directory, you will need to make appropriate changes.

At the c prompt, type

<C:\>mkdir robosim

Change to the new robosim directory by typing

<C:\>cd robosim

Next xcopy all of the files and subdirectories from this diskette by typing

<C:\ROBOSIM>xcopy a:*.* /s/e

Perform a directory listing by typing

C:\ROBOSIM>dir

The directory listing should show the following files and directories:

ROBO386.EXE
GO32.EXE

```
SETUP.CMD
INIT.BAT
PUMA560.KIN
PUMA560.DAT
MANUAL.WP
MANUAL
INSTALL.WP
INSTALL
TUTORIAL.WP
TUTORIAL
README.1ST
FONTS           <DIR>
DRIVERS         <DIR>
EXAMPLE1        <DIR>
EXAMPLE2        <DIR>
EXAMPLE3        <DIR>
EXAMPLE4        <DIR>
EXAMPLE5        <DIR>
EXAMPLE6        <DIR>
EXAMPLE7        <DIR>
EXAMPLE8        <DIR>
EXAMPLE9        <DIR>
EXAMPL10        <DIR>
EXAMPL11        <DIR>
EXAMPL12        <DIR>
EXAMPL13        <DIR>
EXAMPL14        <DIR>
```

The batch file init.bat contains commands that select the graphics driver and specifies the path to the fonts directory. As supplied, the init.bat file is

```
set GO32=driver c:\robosim\drivers\vga16.grd gw 640 gh 480 tw 80 th 25
set GRXFONT=c:\robosim\fonts
```

If you have copied the ROBOSIM files to a different directory than c:\robosim, then you should use an editor to modify the init.bat file accordingly. The driver vga16.grd specified above in the default init.bat file is one of three generic choices. It may or may not work, or be the best choice, on your system. To determine if it will work, type init from the robosim directory, i.e.,

```
<C:\ROBOSIM>init
```

Then type robo386 at the prompt, i.e.,

```
<C:\ROBOSIM>robo386
```

If the vga16.grd driver is appropriate for your system, then you should see a window at the bottom of the screen with the prompt robag>, and a rectangular floor above the window. If this is the case, you may now skip to the section on Operation. If you wish to choose a driver specific to your system, read on.

If the vga16.grd driver is inappropriate for your system, then you may select the proper graphics driver (which must correspond to the graphics adapter in your system) by setting the GO32 environment variable (using your editor) in the init.bat file:

set GO32=driver c:\robosim\drivers\driver.ext gw ??? gh ??? tw ??? th ???

The available drivers are included in the DRIVERS subdirectory. Generic choices are:

| | |
|---|---|
| EGA16.GRD | 16 color EGA 320X200 or 640X350 |
| VGA16.GRD | 16 color VGA 320X200 or 640X350 or 640X480 |
| VGA.GRD | 256 color VGA 320X200 only |

The VGA drivers above should work with any VGA board. Additionally there are board-specific Super VGA drivers in the DRIVERS subdirectory, most of them for 256 colors. (Exception: tseng416.grd which is a 16 color driver.) You will note that all of the driver files contained in the DRIVERS subdirectory have a grd extension except the ATIULTRA.GRN driver file. The gw (graphics width), gh (graphics height), tw (text width), and th (text height) fields can be used to specify the default graphics and text resolutions, respectively.

If you do not know the name of the graphics adapter in your system, you should try different drivers from the DRIVERS subdirectory until you find one that works (starting first with the generic choices listed above). For example, some Gateway 2000 computer systems use the ATIULTRA graphics adapter, so there the GO32 environment variable would be changed in the init.bat file to:

set GO32=driver c:\robosim\drivers\atiultra.grn gw 640 gh 480 tw 80 th 25

This specifies the ATIULTRA.GRN driver file with a graphics resolution of 640X480 pixels and a text resolution of 80X25 lines. These are the default settings for this driver so they could be omitted as in:

set GO32=driver c:\robosim\drivers\atiultra.grn

which would give the same graphics and text resolutions. The ATIULTRA.GRN driver is capable of higher resolutions, including for example 1024X768 graphics resolution. Changing the GO32 environment variable to

set GO32=driver c:\robosim\drivers\atiultra.grn gw 1024 gh 768

will provide the specified higher resolution. Different text resolutions could also be specified. Note: the standard text resolution of 80X25 is recommended, and a graphics resolution of 640X480 is recommended as a good compromise between intensity and resolution.

The second line of the init.bat file specifies the directory where the fonts are installed. If you have installed the ROBOSIM files in a directory other than C:\ROBOSIM, then this line will need to be changed as well as the first line to reflect the different directory that you have chosen.

**Operation:**

Once you have completed the above steps, ROBOSIM should be ready to run on your system. The procedure is to change to the ROBOSIM directory, run the initiate batch file by typing 'init', and type robo386. After startup, the program executes the commands from the file 'setup.cmd' and displays the results on the screen with an interactive window and robag> prompt. The simulator mode may be exited by typing 'exit' at the robag> prompt.

# APPENDIX E:

# EXAMPLE FILES AND GRAPHIC DISPLAY

## EXAMPLE 1: OBJECT CONSTRUCTION

This example contains two files, OBJECT.DAT and SETUP.CMD. The purpose of the example is to show how complex objects may be constructed from solid primitives using the ROBOSIM modeling language. There are two approaches you may use to run the example: one, you may copy the OBJECT.DAT and SETUP.CMD files into your ROBOSIM root directory and run the program from there, or two, you may copy the ROBO386.EXE file from your root directory into this EXAMPLE1 directory, and run the program from this directory. If you use the first approach, you may want to copy the SETUP.CMD file in your root directory to something else so that you can recover it later. If you use the second approach, you will need to run the initiation batch file INIT.BAT from the ROBOSIM root directory where it exists before switching to the EXAMPLE1 directory to run the example. This is necessary to tell the simulator program where the FONTS and DRIVERS directories are located.

An explanation of the program is contained in the comment statements of the OBJECT.DAT and SETUP.CMD files. Before attempting to run the example, you may wish to print a copy of these two files. You will observe in the OBJECT.DAT file there are several 'store-file comp-obj' commands, with all but the last commented out with a semicolon. These commands may be used to view different stages of the object as it is constructed. For example, if you precede the last 'store-file' command with a semicolon (thereby commenting it out) and delete the semicolon in front of the first 'store-file' command, you will observe the single box constructed with the first non-commented command of the file. In a similar manner you may select other store-file commands to view different stages of the object's construction.

To run the program type 'robo386' at the DOS prompt. This will enter the simulator and display whatever precedes the 'store-file' command. You may exit the simulator by typing 'exit' at the robag> prompt. As discussed above, you may then use your editor to select a subsequent 'store-file' command in order to view a later stage of the construction of the composite object.

## FILE: SETUP.CMD

```
;       ************************************************************
;                              SETUP.CMD
;       ************************************************************
;
;          The simulation environment of ROBOSIM is an interactive
;       mode. The commands entered by the user are immediately
;       executed and their results are printed or displayed on the
;       screen. The simulator's code is contained in one executable
```

```
;       file named 'robo386'.  Typing 'robo386' at the shell prompt
;       will invoke it.  Upon startup the simulator initializes the
;       graphics display and loads in initial commands from the
;       file named 'setup.cmd'.  The 'setup.cmd' file, as we will
;       construct here, normally contains as a minimum the color
;       definitions, the view commands, and reference to object
;       and/or robot agent files.
;       ***************************************************************
;
;                       COLOR DEFINITIONS
;       ***************************************************************
define-color red 1.00 0.00 00.0
define-color green 0.00 1.00 0.00
define-color blue 0.00 0.00 1.00
define-color black 0.00 0.00 0.00
define-color purple 1.00 0.00 1.00
define-color cyan 0.00 1.00 1.00
define-color yellow 1.00 1.00 0.00
define-color white 1.00 1.00 1.00
define-color rose 0.70 0.20 0.20
define-color gold 0.50 0.50 0.20
define-color lt-grey 0.75 0.75 0.75
;       ***************************************************************
;
;                       VIEW COMMANDS
;       ***************************************************************
look-at 0 0 700
look-from 5000 0 1000
;       ***************************************************************
;
;               The following command is used to create a composite
;       object using the ROBOSIM modeling environment source code.
;       The specified '.dat' file is scanned until a 'store-file'
;       command is encountered with the specified object name
;       (comp-obj) as its argument.  The composite object is named
;       'object-1' in the simulation environment.  The object
;       'object-1' may be moved, grasped, etc., with the interactive
;       commands of the simulation environment.
;       ***************************************************************
;
make-robosim-object object-1 object.dat comp-obj green
set-collision-check 0
end
```

**FILE: OBJECT.DAT**

```
;       ***************************************************************
;
;                               OBJECT.DAT
;       ***************************************************************
;
```

```
;          The purpose of this program is to demonstrate how
;        objects may be constructed from solid primitives using
;        commands from the ROBOSIM modeling environment.
;          Any convenient editor may be used to assembly the
;        program.  The program file may be named anything you wish
;        with a suffix of dat.  Here I have called the file
;        object.dat.  As can be seen, comments may be added by
;        preceeding them with a semicolon.
;        ***************************************************************
;
box x=200 y=400 z=600
;        ***************************************************************
;
;          The above command stores the graphical and mass
;        properties of a box of the indicated dimensions in the A-
;        register.  The box will be centered at the origin of the
;        display when we view it in the simulation environment.  To
;        prepare the box for display, we store it in a file comp-obj,
;        as done next.  The semicolon must be removed before the
;        store-file command will be executed.  As we construct our
;        composite object, we will be viewing the results as we go
;        along.  For this reason, the program will contain a number
;        of <store-file> commands that have been "commented out" with
;        the semicolon, once we proceed beyond that point in the
;        program.
;        ***************************************************************
;
; store-file comp-obj
;        ***************************************************************
;
;          To view the box we must prepare a setup.cmd file that
;        includes the simulation enviroment command <make-robosim-
;        object>, as described elsewhere.  Then when we enter the
;        simulation environment by typing robo386, the box will be
;        displayed at the origin.
;        ***************************************************************
;
store b
;        ***************************************************************
;
;          The above command stores the contents of the A-register
;        in the B-register.
;        ***************************************************************
;
clear
;        ***************************************************************
;
;          The <clear> command clears the A-register.
;        ***************************************************************
;
cone r=100 h=500
;        ***************************************************************
;
;          The preceeding command stores the graphical and mass
;        properties of a cone of the indicated dimensions in the A-
```

```
;       register.  The cone will be centered at the origin at its
;       center of mass.
;       ***************************************************************
; store-file comp-obj
translate x=0 y=0 z=467
; store-file comp-obj
add b
;       ***************************************************************
;           The above command adds the contents of the B-register
;       (box constructed previously) to the A-register.  Now, when
;       viewed, our composite object will consist of the box with
;       the cone attached to its top.
;       ***************************************************************
store b
; store-file comp-obj
clear
vector x=0 y=300 z=0
;       ***************************************************************
;           The preceeding command draws a line from the origin to
;       the point specified.  Note:  If we were to try to view the
;       single line, it would not be displayed.  The reason is that
;       the present version of ROBOSIM checks all objects (the
;       single line is an object) for perpendicular directions to
;       polygon surfaces.  Since a single line does not define a
;       polygon, the line is simply not displayed.
;       ***************************************************************
; store-file comp-obj
translate x=0 y=100 z=0
;       ***************************************************************
;           The above command translates the line 100 units along
;       the y-axis.
;       ***************************************************************
draw x=0 y=400 z=400
;       ***************************************************************
;           The draw command generates a visible vector from the
;       previous vector tip to the point specified.
;       ***************************************************************
draw x=0 y=200 z=300
draw x=0 y=100 z=0
; store-file comp-obj
rev-surface
; store-file comp-obj
translate x=0 y=0 z=800
add b
; store-file comp-obj
```

```
rotate x=45
store-file comp-obj
end
```

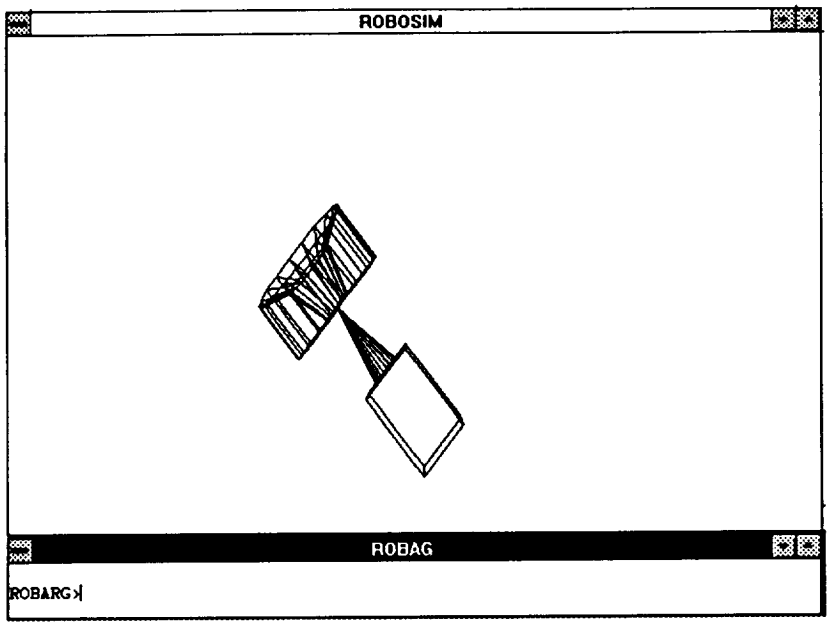


Figure E.1. Sample display for Example 1.

## EXAMPLE 2: FRAME ROTATIONS

The files in this example will produce 2 identically labelled frames (frame1 and frame2) that are useful for demonstrating rotations of one frame about another.

To run the example, you must first load the ROBO386.EXE file from your ROBOSIM root directory into this directory. You must also run the initiation batch file INIT.BAT from the root directory in order to tell the simulator program where the FONTS and DRIVERS directories are located. To run the program type 'robo386' from the DOS prompt in this directory. Use the command 'rotate-object frame1 x=X y=Y z=Z' (X, Y, and Z in degrees) to rotate frame1 relative to frame2.

The 'look-at' and 'look-from' commands are initially set in the SETUP.CMD file as,

```
look-at 0 0 0
look-from 0 500 0
```

You may wish to change the 'look-from' command interactively from the robag> prompt to obtain a better viewing position for the rotations you make. You may exit the simulator mode by typing 'exit' at the robag> prompt. This will return you to the DOS prompt.

## FILE: SETUP.CMD

```
;       ***************************************************************
;                              SETUP.CMD
;       ***************************************************************
;                          COLOR DEFINITIONS
;       ***************************************************************
define-color red 1.00 0.00 00.0
define-color green 0.00 1.00 0.00
define-color blue 0.00 0.00 1.00
define-color black 0.00 0.00 0.00
define-color purple 1.00 0.00 1.00
define-color cyan 0.00 1.00 1.00
define-color yellow 1.00 1.00 0.00
define-color white 1.00 1.00 1.00
define-color rose 0.70 0.20 0.20
define-color gold 0.50 0.50 0.20
define-color lt-grey 0.75 0.75 0.75
;       ***************************************************************
;                          VIEW COMMANDS
;       ***************************************************************
look-at 0 0 0
look-from 0 500 0
;       ***************************************************************
;                          LOAD FILE:
;       ***************************************************************
load 2-frames
set-collision-check 0
end
```

**FILE: X-AXIS.DAT**

```
clear
move x=-100 y=0 z=0
draw x=100 y=0 z=0
draw x=100 y=0 z=5
store b
clear
move x=.1 y=0 z=100
draw x=5 y=0 z=100
draw x=.1 y=0 z=110
rev-surface
rotate y=90
add b
store b
clear
```

```
move x=123 y=0 z=5
draw x=117 y=0 z=-5
draw x=117 y=0 z=-5.01
move x=123 y=0 z=-5.01
draw x=117 y=0 z=5
draw x=117 y=0 z=5.01
add b
store-file x-axis
end
```

**FILE: Y-AXIS.DAT**

```
clear
move x=-100 y=0 z=0
draw x=100 y=0 z=0
draw x=100 y=0 z=5
store b
clear
move x=.1 y=0 z=100
draw x=5 y=0 z=100
draw x=.1 y=0 z=110
rev-surface
rotate y=90
add b
rotate z=90
store b
clear
move x=0 y=123 z=-5
draw x=0 y=117 z=5
draw x=0 y=117 z=5.01
move x=0 y=120 z=0
draw x=0 y=123 z=5
draw x=0 y=123 z=5.01
add b
store-file y-axis
end
```

**FILE: Z-AXIS.DAT**

```
clear
move x=-100 y=0 z=0
draw x=100 y=0 z=0
draw x=100 y=0 z=5
store b
clear
```

```
move x=.1 y=0 z=100
draw x=5 y=0 z=100
draw x=.1 y=0 z=110
rev-surface
rotate y=90
add b
rotate y=-90
store b
clear
move x=-3 y=0 z=120
draw x=3 y=0 z=120
draw x=-3 y=0 z=130
draw x=3 y=0 z=130
add b
store-file z-axis
end
```

**FILE: 2-FRAMES**

```
make-robosim-object x-axis x-axis.dat x-axis red
make-robosim-object y-axis y-axis.dat y-axis white
make-robosim-object z-axis z-axis.dat z-axis blue
link-objects frame1 x-axis y-axis z-axis
make-robosim-object x-axis x-axis.dat x-axis red
make-robosim-object y-axis y-axis.dat y-axis white
make-robosim-object z-axis z-axis.dat z-axis blue
link-objects frame2 x-axis y-axis z-axis
exit
```

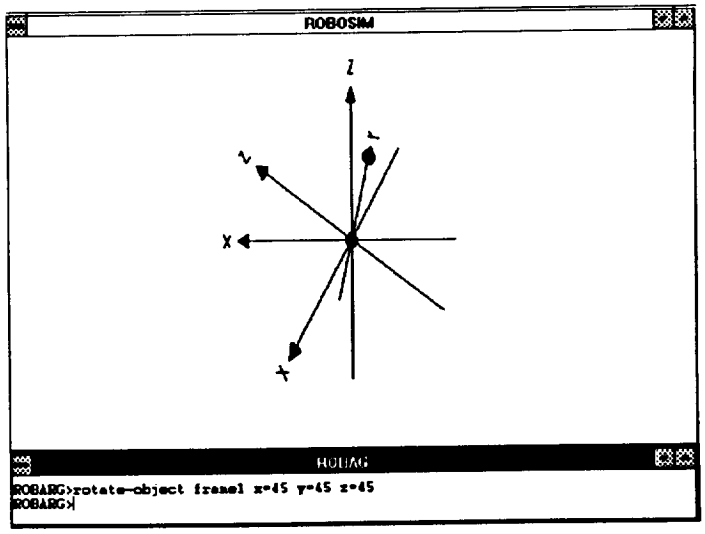


Figure E.2. Sample display for Example 2.

## EXAMPLE 3: CONSTRUCTION OF ROBOT AND ASSIGNMENT OF LINKS

The objective of this example is to provide a brief tutorial on constructing a robot and assigning the link joint frames. The reader is encouraged to print a copy of the ARM.DAT file where a thorough discussion is given of the ROBOSIM modeler commands for doing this.

In order to run this example, you must copy the ROBO386.EXE file from your ROBOSIM root directory into this directory. You must also run the initiation batch file INIT.BAT from the root directory to tell the simulator program where the FONTS and DRIVERS directories are located.

To view the robot constructed in the ARM.DAT file type 'robo386' at the DOS prompt. As explained in the ARM.DAT file comments, you may move the robot with the 'drive' command. For example,

robag>drive 2-axes-r 0 360 0 0 0 0

will rotate the second joint of the robot (called "2-axes-r") 360 degrees.

You may run the DEMO.CMD file by typing 'load demo.cmd' at the robag> prompt. You may exit the simulator mode by typing 'exit' at the robag> prompt.

## FILE: SETUP.CMD

```
;       ************************************************************
;                              SETUP.CMD
;       ************************************************************
;          The simulation environment of ROBOSIM is an interactive
;       mode. The commands entered by the user are immediately
;       executed and their results are printed or displayed on the
;       screen. The simulator's code is contained in one executable
;       file named 'robo386'. Typing 'robo386' at the shell prompt
;       will invoke it. Upon startup the simulator initializes the
;       graphics display and loads in initial commands from the
;       file named 'setup.cmd'. The 'setup.cmd' file, as we will
;       construct here, normally contains as a minimum the color
;       definitions, the view commands, and reference to object
;       and/or robot agent files.
;       ************************************************************
;                          COLOR DEFINITIONS
;       ************************************************************
define-color red 1.00 0.00 00.0
define-color green 0.00 1.00 0.00
```

```
define-color blue 0.00 0.00 1.00
define-color black 0.00 0.00 0.00
define-color purple 1.00 0.00 1.00
define-color cyan 0.00 1.00 1.00
define-color yellow 1.00 1.00 0.00
define-color white 1.00 1.00 1.00
define-color rose 0.70 0.20 0.20
define-color gold 0.50 0.50 0.20
define-color lt-grey 0.75 0.75 0.75
;       ************************************************************
;
;                       VIEW COMMANDS
;       ************************************************************
look-at 0 0 500
look-from 4000 0 500
;       ************************************************************
;
;               The following command is used to create a robot whose
;       model has been pre-compiled using the ROBOSIM modeling
;       commands.  The agent-type is 'arm'.  This is the 'arm.dat'
;       file where the robot (agent) is modelled.  The 'dat' suffix
;       is not included in the agent-type specification.  Here we
;       have named the robot '2-axes-r'.  The 2-axes robot '2-axes-r'
;       can be moved about the environment with the 'translate-object'
;       or 'rotate-object' commands.  The two axes of the robot may
;       be driven with the 'drive' command.  The 'drive' command takes
;       six joint angles.  For our 2-axes robot, the last 4 joint
;       angles are zero.
;       ************************************************************
make-agent 2-axes-r arm green
make-object box1 box 100 100 150 cyan
define-grasping-point box1 0 0 0 0 0 0
translate-object box1 y=-550 z=75
make-object box2 box 100 100 150 yellow
define-grasping-point box2 0 0 0 0 0 0
translate-object box2 x=550 z=75
make-object box3 box 100 100 150 white
define-grasping-point box3 0 0 0 0 0 0
translate-object box3 y=550 z=75
set-collision-check 0
end
```

**FILE: ARM.DAT**

```
;       ************************************************************
;
;
;                               ARM.DAT
```

```
;
;          CONSTRUCTION OF TWO-AXIS MANIPULATOR ARM
;
;*************************************************************
; INTRODUCTION:
;     Most common robots are composed of a series of rigid
; components or links that are connected by rotary (revolute)
; joints or sliding (prismatic) joints. The number of joints
; (degrees-of-freedom) and their placement with respect to one
; another greatly affect the volume (workspace) through which
; the last link of the mechanism (end-effector) can be
; translated and the directions along which the end-effector
; may be oriented. Mechanisms with a form consisting of rigid
; links connected end-to-end are called serial kinematic chains
; and constitute the vast majority of robots in use today.
; Other types of mechanisms, e.g., parallel linkages, consist
; of linkage elements that are connected in parallel. Examples
; of this type of mechanism are four bar linkages and six DOF
; motion-base platforms similar to that used in cockpit flight
; simulators. ROBOSIM at the present time does not support
; experimentation with parallel linkages.
;     Modeling a serial linkage arm with ROBOSIM is
; accomplished by using the geometric modeling commands to
; develop a replica of each link of the robot mechanism from
; geometric primitives. In addition to the shape of each link
; the joint locations and types must be specified to establish
; the connective relationships of adjacent links. The types of
; joints that are permitted are: R-joints for revolute; P-joints
; for prismatic; and F-joints for fixed joints. R-joints and
; P-joints are used to specify the moveable joints while the
; F-joint or fixed joint allows us to rigidly attach fixed parts
; to a robot link element, e.g., the end-of-arm tooling. The
; use of the F-joint type also allows us to regard all link
; elements in the kinematic chain in a general way, thus each
; link will have an input joint (I) and an output joint (I+1)
; which may be either R, P, or F-type. When the arm is viewed
; as a complete assembly, the output joint (I+1) on one link is
; connected to the input joint (I) of the next link (moving
; toward the end-effector). To be consistent, these common
; joints located in adjacent links must be of the same type
; whether R, P, or F-type.
;*************************************************************
; SET UP WORKCELL COORDINATE FRAMES:
;     The first element of our robot system to be specified
; is the workcell. The workcell is regarded the same logically
```

```
;       as any link in the kinematic chain. The workcell reference
;       frame is specified with a fixed joint type. To be consistent
;       with links of the robot, two fixed joint types are used to
;       specify the workcell frame. The fixed joints, F-JOINT-I and
;       F-JOINT-I+1, used to specify the workcell frame may be located
;       anywhere. Here both joints will be located at the origin.
;       The first joint of the base link (l0) will be aligned with the
;       F-JOINT-I+1 of the workcell. The workcell, with joints attached,
;       must be stored as the first "link" of the robot system. It is
;       always designated with a <.LOC> suffix.
;       ************************************************************
;
f-joint-i
f-joint-i+1
store-link arm.loc
;       ************************************************************
;          In the above commands, the F-JOINT-I command causes
;       ROBOSIM to define a fixed input joint at the current origin in
;       the A-register. This fixed joint or attach point is the first
;       in the series that will ultimately end at the robot's end-
;       effector tool center point (TCP). The F-JOINT-I+1 causes
;       ROBOSIM to define a fixed output joint superimposed on the
;       fixed input joint at the origin in A-register. It is not
;       necessary that the fixed input and output joints of the workcell
;       be superimposed; they are superimposed here for convenience.
;       When the assembled robot is viewed in the simulation
;       environment of ROBOSIM, the input joint of the robot's base
;       link will be aligned with the output joint of the workcell.
;       The definition of the workcell is completed with the STORE-LINK
;       command. The argument <ARM.LOC> specifies that robot's name
;       is ARM, and the extension <.LOC> indicates that the information
;       specifies the workcell's geometry.
;       ************************************************************
;       BASE LINK:
;          The first joint of the base link will be aligned with the
;       fixed joint, F-JOINT-I+1, of the workcell frame. The first
;       joint of the base link should be a fixed joint to match the
;       fixed I+1-joint of the workcell. The I+1-joint of the base
;       link is the 1st moveable joint of the manipulator. It must
;       match the I-joint of the next link of the manipulator.
;       ************************************************************
;
clear
cylinder r=200 h=100
store b
clear
move x=200 y=0 z=50
```

```
draw x=100 y=0 z=400
rev-surface
add b
store b
clear
cylinder r=100 h=100
translate x=0 y=0 z=450
add b
store b
clear
f-joint-i
translate x=0 y=0 z=-50
add b
store b
clear
r-joint-i+1
translate x=0 y=0 z=500
add b
store-link arm.l0
;       ****************************************************************
;       FIRST LINK:
;       ****************************************************************
clear
cylinder r=100 h=100
store b
clear
move x=50 y=0 z=0
draw x=50 y=100 z=0
draw x=25 y=125 z=0
draw x=-25 y=125 z=0
draw x=-50 y=100 z=0
draw x=-50 y=0 z=0
draw x=50 y=0 z=0
extrude-surface z=600
rotate x=90
rotate z=90
translate x=200 y=0 z=50
add b
store b
clear
r-joint-i
rotate z=90
translate x=0 y=0 z=-50
add b
store b
```

```
clear
r-joint-i+1
rotate y=90
rotate x=90
translate x=500 y=0 z=100
add b
store-link arm.l1
;       ****************************************************************
;
;       SECOND LINK:
;       ****************************************************************
;
clear
move x=50 y=0 z=0
draw x=50 y=100 z=0
draw x=25 y=125 z=0
draw x=-25 y=125 z=0
draw x=-50 y=100 z=0
draw x=-50 y=0 z=0
draw x=50 y=0 z=0
extrude-surface z=100
store b
clear
box x=100 y=500 z=100
translate x=0 y=-250 z=0
add b
store b
clear
r-joint-i
rotate x=180
rotate z=180
translate x=0 y=50 z=50
add b
store b
clear
r-joint-i+1
translate x=0 y=50 z=0
add b
store-link arm.l2
end
```

**FILE: DEMO.CMD**

```
minimal-step 1
drive 2-axes-r 360 0 0 0 0 0
minimal-step 10
grasp 2-axes-r box1
```

```
drive 2-axes-r 0 55 0 0 0 0
drive 2-axes-r 135 55 0 0 0 0
drive 2-axes-r 135 0 0 0 0 0
release 2-axes-r box1
drive 2-axes-r 180 0 0 0 0 0
grasp 2-axes-r box3
drive 2-axes-r 180 -55 0 0 0 0
drive 2-axes-r 45 -55 0 0 0 0
drive 2-axes-r 45 0 0 0 0 0
release 2-axes-r box3
drive 2-axes-r 90 0 0 0 0 0
grasp 2-axes-r box2
drive 2-axes-r 90 55 0 0 0 0
drive 2-axes-r 225 55 0 0 0 0
drive 2-axes-r 225 0 0 0 0 0
release 2-axes-r box2
drive 2-axes-r 45 0 0 0 0 0
grasp 2-axes-r box3
drive 2-axes-r 45 55 0 0 0 0
drive 2-axes-r 180 55 0 0 0 0
drive 2-axes-r 180 0 0 0 0 0
release 2-axes-r box3
drive 2-axes-r 225 0 0 0 0 0
grasp 2-axes-r box2
drive 2-axes-r 225 -55 0 0 0 0
drive 2-axes-r 90 -55 0 0 0 0
drive 2-axes-r 90 0 0 0 0 0
release 2-axes-r box2
drive 2-axes-r 135 0 0 0 0 0
grasp 2-axes-r box1
drive 2-axes-r 135 55 0 0 0 0
drive 2-axes-r 270 55 0 0 0 0
drive 2-axes-r 270 0 0 0 0 0
release 2-axes-r box1
drive 2-axes-r 90 0 0 0 0 0
grasp 2-axes-r box2
drive 2-axes-r 90 55 0 0 0 0
drive 2-axes-r 225 55 0 0 0 0
drive 2-axes-r 225 0 0 0 0 0
release 2-axes-r box2
drive 2-axes-r 270 0 0 0 0 0
grasp 2-axes-r box1
drive 2-axes-r 270 -55 0 0 0 0
drive 2-axes-r 135 -55 0 0 0 0
drive 2-axes-r 135 0 0 0 0 0
```

```
release 2-axes-r box1
drive 2-axes-r 180 0 0 0 0 0
grasp 2-axes-r box3
drive 2-axes-r 180 55 0 0 0 0
drive 2-axes-r 315 55 0 0 0 0
drive 2-axes-r 315 0 0 0 0 0
release 2-axes-r box3
drive 2-axes-r 135 0 0 0 0 0
grasp 2-axes-r box1
drive 2-axes-r 135 55 0 0 0 0
drive 2-axes-r 270 55 0 0 0 0
drive 2-axes-r 270 0 0 0 0 0
release 2-axes-r box1
drive 2-axes-r 315 0 0 0 0 0
grasp 2-axes-r box3
drive 2-axes-r 315 -55 0 0 0 0
drive 2-axes-r 180 -55 0 0 0 0
drive 2-axes-r 180 0 0 0 0 0
release 2-axes-r box3
drive 2-axes-r 225 0 0 0 0 0
grasp 2-axes-r box2
drive 2-axes-r 225 55 0 0 0 0
drive 2-axes-r 0 55 0 0 0 0
drive 2-axes-r 0 0 0 0 0 0
release 2-axes-r box2
minimal-step 1
drive 2-axes-r 360 0 0 0 0 0
end
```

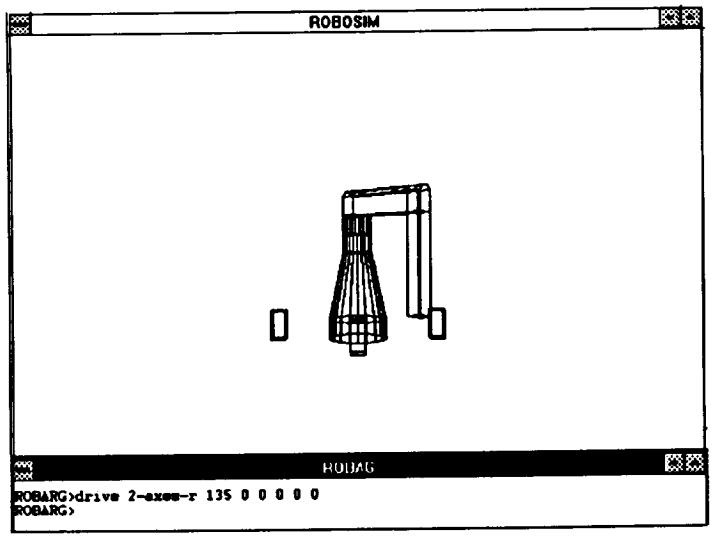


Figure E.3. Sample display for Example 3.

## EXAMPLE 4: ROBOT REMOVAL/INSERTION OF MODULE

This example demonstrates a robot removing and inserting a module in different configurations of the arm. To run the program you should first copy the ROBO386.EXE file from your ROBOSIM root directory to this directory. You will also need to execute the initiation batch file INIT.BAT so that the program will know the path to the FONTS and DRIVERS directories.

To run the demonstration, type 'robo386' at the DOS prompt, and then 'load demo.cmd' at the simulator's robag> prompt. The program will execute the removal and insertion routines on both sides of the robot in the elbow-up and elbow-down configurations. When the demo is completed the program will return to the simulator's robag> prompt.

You can observe the system's collision detection and avoidance heuristics by lowering the top object to cause a collision. To do this type 'translate-object top z=-250' at the robag> prompt. Now when you run the program by typing 'load demo.cmd', a collision will be detected in attempting to perform part of the routines on the right hand side of the display. Once a collision is detected, the heuristics will search for a collision-free path. When a complete collision-free path is found, the robot will return quickly to the start position and then execute the complete collision-free path that has been determined.

To exit the simulator mode type 'exit' at the robag> prompt.

## FILE: SETUP.CMD

```
define-color red 1 0 0
define-color green 0 1 0
define-color blue 0 0 1
define-color black 0 0 0
define-color purple 1 0 1
define-color cyan 0 1 1
define-color yellow 1 1 0
define-color white 1 1 1
look-at 0 0 700
look-from 0 5000 1000
make-agent r puma560 red
make-object floor box 2000 2000 60 green
translate-object floor z=-31
make-object xx box 200 10 100 blue
define-grasping-point xx -180 0 0 -90 0 -90
translate-object xx x=900 y=300 z=800
make-object top cylinder 200 10 blue
```

```
translate-object top z=1650
set-collision-check 1
end
```

**FILE: DEMO.CMD**

```
set-echo 1
set-solution r 0
load moveback.cmd
load moveforw.cmd
set-solution r 3
load moveback.cmd
load moveforw.cmd
rotate-object xx z=180
translate-object xx x=200
set-solution r 4
load moveforw.cmd
load moveback.cmd
set-solution r 7
load moveforw.cmd
load moveback.cmd
translate-object top z=-250
set-solution r 7
load moveforw.cmd
drive r 0 0 0 0 0 100
end
```

**FILE: MOVEFORW.CMD**

```
move-to-grasp r xx
get-solution r
grasp r xx
translate-agent r x=-200
translate-agent r y=600
translate-agent r x=200
release r xx
drive-find r 0 0 0 0 0 0
end
```

**FILE: MOVEBACK.CMD**

```
move-to-grasp r xx
get-solution r
grasp r xx
translate-agent r x=-200
```

```
translate-agent r y=-600
translate-agent r x=200
release r xx
drive-find r 0 0 0 0 0 0
end
```

**FILE: PUMA560.DAT**

```
ECHO OFF
;*************************************************
;
;* VANDERBILT UNIVERSITY
;* 1, AUG 1988
;* MODELLING THE  PUMA-560
;* SALEH ZEIN
;* Modified by Csaba Biegl for Boeing
;*************************************************
;
SET-NFACE N=25.0
;*************************************************
;
; SET UP  CELL COORDINATE FRAMES
;*************************************************
;
CLEAR
F-JOINT-I
F-JOINT-I+1
STORE-LINK PUMA560.LOC
;*************************************************
;
; BASE LINK-L0
;*************************************************
;
CLEAR
ADD B
ADD C
CYLINDER R=88.90,H=533.40
TRANSLATE X=0.00,Y=0.00,Z=328.93
STORE B
CLEAR
CYLINDER R=177.80,H=25.40
TRANSLATE X=0.00,Y=0.00,Z=49.53
ADD B
STORE B
CLEAR
CYLINDER R=203.20,H=36.83
TRANSLATE X=0.00,Y=0.00,Z=18.41
F-JOINT-I
ADD B
STORE B
CLEAR
```

```
CYLINDER R=38.10,H=152.40
TRANSLATE X=0.00,Y=0.00,Z=127.00
BOX X=127.00,Y=101.60,Z=101.60
TRANSLATE X=0.00,Y=-152.40,Z=50.80
ADD B
STORE B
CLEAR
R-JOINT-I+1
TRANSLATE X=0.00,Y=0.00,Z=671.83
ADD B
STORE-LINK PUMA560.L0
.*********************************************
;
; FIRST LINK-L1
.*********************************************
;
CLEAR
STORE B
CYLINDER R=88.90,H=25.40
TRANSLATE X=0.00,Y=0.00,Z=25.40
CYLINDER R=76.20,H=25.40
TRANSLATE X=0.00,Y=0.00,Z=116.84
STORE B
CLEAR
CYLINDER R=68.50,H=232.68
TRANSLATE X=0.00,Y=0.00,Z=-15.0
ADD B
TRANSLATE X=0.00,Y=0.00,Z=32.64
STORE B
CLEAR
R-JOINT-I+1
ADD B
STORE B
CLEAR
R-JOINT-I
ROTATE X=90.
ADD B
STORE-LINK PUMA560.L1
.*********************************************
;
; SECOND LINK-L2
.*********************************************
;
CLEAR
STORE B
STORE C
MOVE X=0.00,Y=-228.60,Z=0.00
DRAW X=0.00,Y=-226.82,Z=-31.75
DRAW X=0.00,Y=-216.66,Z=-83.82
```

```
DRAW X=0.00,Y=-200.66,Z=-127.00
DRAW X=0.00,Y=50.80,Z=-127.00
DRAW X=0.00,Y=431.80,Z=-76.20
DRAW X=0.00,Y=451.61,Z=-73.66
DRAW X=0.00,Y=469.90,Z=-66.04
DRAW X=0.00,Y=485.65,Z=-53.85
DRAW X=0.00,Y=497.84,Z=-38.10
DRAW X=0.00,Y=505.46,Z=-19.81
DRAW X=0.00,Y=508.00,Z=0.00
DRAW X=0.00,Y=505.46,Z=19.81
DRAW X=0.00,Y=497.84,Z=38.10
DRAW X=0.00,Y=485.65,Z=53.85
DRAW X=0.00,Y=469.90,Z=66.04
DRAW X=0.00,Y=451.61,Z=73.66
DRAW X=0.00,Y=431.80,Z=76.20
DRAW X=0.00,Y=50.80,Z=127.00
DRAW X=0.00,Y=-200.66,Z=127.00
DRAW X=0.00,Y=-216.66,Z=83.82
DRAW X=0.00,Y=-226.82,Z=31.75
DRAW X=0.00,Y=-228.60,Z=0.00
ROTATE Y=90.
EXTRUDE-SURFACE Z=101.60
ROTATE Z=-90.
TRANSLATE X=0.00,Y=0.00,Z=243.08
STORE C
CLEAR
R-JOINT-I
ADD C
STORE C
CLEAR
R-JOINT-I+1
ROTATE Z=-180.
TRANSLATE X=431.80,Y=0.00,Z=149.10
ADD C
STORE-LINK PUMA560.L2
;*****************************************
;
;  LINK-L3
;*****************************************
;
CLEAR
STORE B
STORE C
MOVE X=0.00,Y=-127.00,Z=0.00
DRAW X=0.00,Y=-124.97,Z=-22.10
DRAW X=0.00,Y=-122.68,Z=-32.77
DRAW X=0.00,Y=-119.38,Z=-43.43
```

```
DRAW X=0.00,Y=-113.54,Z=-57.15
; DRAW X=0.00,Y=350.52,Z=-44.45
; DRAW X=0.00,Y=350.52,Z=0.00
; DRAW X=0.00,Y=350.52,Z=44.45
DRAW X=0.00,Y=350.52,Z=-14.45
DRAW X=0.00,Y=350.52,Z=30.00
DRAW X=0.00,Y=350.52,Z=74.45
DRAW X=0.00,Y=-113.54,Z=57.15
DRAW X=0.00,Y=-119.38,Z=43.43
DRAW X=0.00,Y=-122.68,Z=32.77
DRAW X=0.00,Y=-124.97,Z=22.10
DRAW X=0.00,Y=-127.00,Z=0.00
TRANSLATE X=0.00,Y=0.00,Z=-30.00
ROTATE Y=90.
EXTRUDE-SURFACE Z=88.90
STORE C
CLEAR
R-JOINT-I+1
ROTATE X=-90.
TRANSLATE X=0.00,Y=433.07,Z=0.00
ADD C
STORE C
CLEAR
R-JOINT-I
TRANSLATE X=-20.32,Y=0.00,Z=0.00
ADD C
STORE-LINK PUMA560.L3
;*****************************************
;
; LINK-L4  WRIST ROTATION
;*****************************************
;
CLEAR
STORE B
STORE C
BOX X=88.90,Y=88.90,Z=6.35
TRANSLATE X=0.00,Y=0.00,Z=15.88
BOX X=88.90,Y=88.90,Z=25.40
TRANSLATE X=0.00,Y=0.00,Z=19.05
BOX X=88.90,Y=88.90,Z=12.70
TRANSLATE X=0.00,Y=0.00,Z=6.35
STORE B
CLEAR
CYLINDER R=40.64,H=30.48
ROTATE X=90.
TRANSLATE X=0.00,Y=29.21,Z=85.09
STORE C
```

```
ADD B
STORE B
CLEAR
LOAD C
TRANSLATE X=0.00,Y=-58.42,Z=0.00
ADD B
TRANSLATE X=0.00,Y=0.00,Z=-85.09
STORE B
CLEAR
R-JOINT-I
ADD B
STORE B
CLEAR
R-JOINT-I+1
ROTATE X=90.
ADD B
STORE-LINK PUMA560.L4
;***********************************************
;
; LINK-L5
;***********************************************
;
CLEAR
STORE B
CYLINDER R=40.64,H=25.40
R-JOINT-I
STORE B
CLEAR
CYLINDER R=25.40,H=5.08
TRANSLATE X=0.00,Y=0.00,Z=7.62
CYLINDER R=12.70,H=10.16
TRANSLATE X=0.00,Y=0.00,Z=5.08
ROTATE X=-90.
TRANSLATE X=0.00,Y=40.64,Z=0.00
ADD B
STORE B
CLEAR
R-JOINT-I+1
ROTATE X=-90.
TRANSLATE X=0.00,Y=55.88,Z=0.00
ADD B
STORE-LINK PUMA560.L5
;***********************************************
;
; LINK-L6
;***********************************************
;
CLEAR
STORE B
```

```
F-JOINT-I+1
STORE B
CLEAR
R-JOINT-I
ADD B
STORE-LINK PUMA560.L6
;*****************************************
; create the object as a fixed joint
;*****************************************
CLEAR
F-JOINT-I
F-JOINT-I+1
STORE-LINK PUMA560.L7
END
```

**FILE: PUMA560.KIN**

```
;;
;; Auxilary kinematics information for the PUMA 560 robot
;;

;
; LINK PARAMETERS
;  This command can be used to overwrite the ROBOSIM-generated
;  link parameters in the model after loading. It is necessary
;  because ROBOSIM does not follow the Denavit-Hartenberg
;  convention. We don't use it here, since the built-in
;  PUMA inverse kinematics routine has been modified to
;  accept ROBOSIM's conventions.
;              JOINT THETA      DZ     DA     ALPHA
; linkparam    1     0.0        0.0    0.0    -90.0
; linkparam    2     0.0        149.5  432.0  0.0
; linkparam    3     0.0        0.0    20.3   90.0
; linkparam    4     0.0        432.0  0.0    -90.0
; linkparam    5     0.0        0.0    0.0    90.0
; linkparam    6     0.0        56.5   0.0    0.0

;
; JOINT LIMITS
;              JOINT MIN    MAX
jointlimit     1     -160.0 160.0
jointlimit     2     -225.0 45.0
jointlimit     3     -45.0  225.0
jointlimit     4     -110.0 170.0
jointlimit     5     -100.0 100.0
```

```
jointlimit      6       -266.0 266.0

;
; ANALYTICAL INVERSE KINEMATICS SOLUTION
;
invertfunc      puma560_inv
```

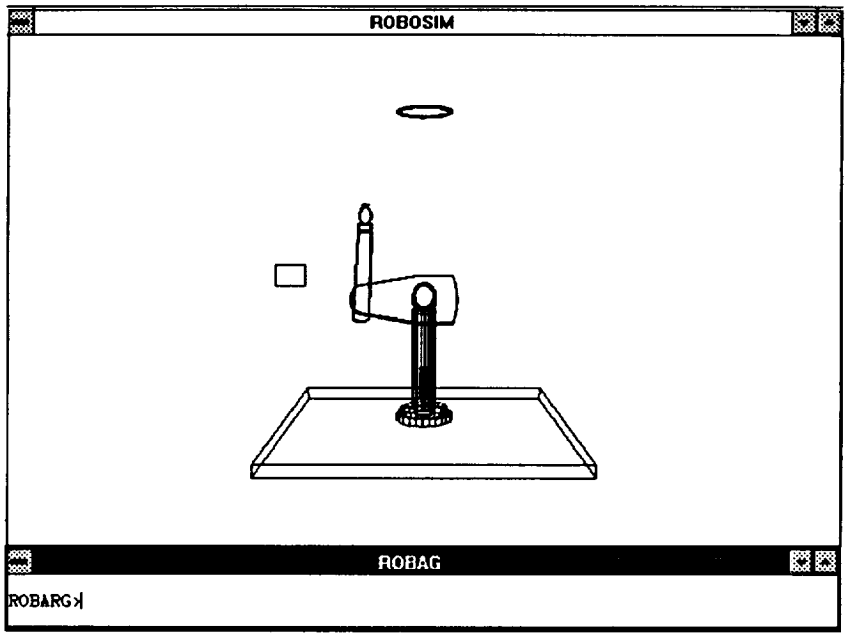


Figure E.4. Sample display for Example 4.

## EXAMPLE 5: CONSTRUCTION OF 3-REVOLUTE JOINT ROBOT

This example shows the construction of a 3-revolute joint robot and demonstrates it stacking blocks.

The reader is encouraged to study the 3R-ROBOT.DAT, SETUP.CMD, and DEMO.CMD files for examples of constructing a robot, assigning link joint frames, and performing tasks with the robot. To view the robot and demo, you must first copy the ROBO386.EXE file from your DOS root directory to this directory. You must also first run the initiation batch file INIT.BAT from the root directory to tell the simulator program where the FONTS and DRIVERS directories are located.

To view the robot constructed in the 3R-ROBOT.DAT file, type 'robo386' at the DOS prompt. To run the demo, type 'load demo.cmd' at the robag> prompt. When the demo is completed, you may exit the simulator mode by typing 'exit' at the robag> prompt.

The robot in this example is called "3r". You may operate it from the interactive window of the simulator by use of the 'drive' command. For example,

robag>drive 3r 90 0 0 0 0 0

will rotate the first joint by 90 degrees. Even though this robot only has 3 joint, all six arguments of the drive command must be specified. The last three, of course, have not effect on this robot.

**FILE: SETUP.CMD**

```
;       ***************************************************************
;                               SETUP.CMD
;       ***************************************************************
;                           COLOR DEFINITIONS
;       ***************************************************************
define-color red 1.00 0.00 00.0
define-color green 0.00 1.00 0.00
define-color blue 0.00 0.00 1.00
define-color black 0.00 0.00 0.00
define-color purple 1.00 0.00 1.00
define-color cyan 0.00 1.00 1.00
define-color yellow 1.00 1.00 0.00
define-color white 1.00 1.00 1.00
define-color rose 0.70 0.20 0.20
define-color gold 0.50 0.50 0.20
define-color lt-grey 0.75 0.75 0.75
;       ***************************************************************
;                           VIEW COMMANDS
;       ***************************************************************
look-at 0 0 750
look-from 0 4000 750
;       ***************************************************************
;                           HOME POSITION VECTOR
;       ***************************************************************
load-positions h
;       ***************************************************************
;                       CREATE ROBOT
;       ***************************************************************
make-agent 3r 3r-robot cyan
;       ***************************************************************
;                       CREATE BLOCKS TO BE MOVED AND STACKED BY ROBOT
;       ***************************************************************
make-object box1 box 600 150 100 red
define-grasping-point box1 0 0 0 0 0 0
translate-object box1 x=-3000
make-object box2 box 600 150 100 white
define-grasping-point box2 0 0 0 0 0 0
```

```
translate-object box2 x=-3000
make-object box3 box 600 150 100 blue
define-grasping-point box3 0 0 0 0 0 0
translate-object box3 x=-3000
make-object box4 box 600 150 100 yellow
define-grasping-point box4 0 0 0 0 0 0
translate-object box4 x=-3000
make-object box5 box 600 150 100 green
define-grasping-point box5 0 0 0 0 0 0
translate-object box5 x=-3000
make-object box6 box 600 150 100 cyan
define-grasping-point box6 0 0 0 0 0 0
translate-object box6 x=-3000
make-object box7 box 600 150 100 purple
define-grasping-point box7 0 0 0 0 0 0
translate-object box7 x=-3000
make-object box8 box 600 150 100 white
define-grasping-point box8 0 0 0 0 0 0
translate-object box8 x=-3000
make-object box9 box 600 150 100 blue
define-grasping-point box9 0 0 0 0 0 0
translate-object box9 x=-3000
make-object box10 box 600 150 100 red
define-grasping-point box10 0 0 0 0 0 0
translate-object box10 x=-3000
set-collision-check 0
end
```

**FILE: 3R-ROBOT.DAT**

```
;       ****************************************
;               3R-ROBOT.DAT
;       ****************************************
;       MODELING OF ROBOT WITH 3 REVOLUTE JOINTS
;       ****************************************
;       WORKCELL COORDINATE FRAMES
;       ****************************************
f-joint-i
f-joint-i+1
store-link 3r.loc
;       ****************************************
;       BASE LINK
;       ****************************************
clear
cylinder r=185 h=50
```

```
store b
clear
cylinder r=40 h=100
translate x=125 y=0 z=75
add b
store b
clear
cylinder r=75 h=250
translate x=0 y=0 z=150
add b
store b
clear
f-joint-i
translate x=0 y=0 z=-25
add b
store b
clear
r-joint-i+1
translate x=0 y=0 z=275
add b
store b
store-link 3r.l0
;       ****************************************
;       FIRST LINK
;       ****************************************
clear
cylinder r=75 h=500
store b
clear
r-joint-i
translate x=0 y=0 z=-250
add b
store b
clear
r-joint-i+1
rotate x=90
translate x=0 y=0 z=200
add b
store b
store-link 3r.l1
;       ****************************************
;       SECOND LINK
;       ****************************************
clear
move x=150 y=0 z=0
```

```
draw x=150 y=-80 z=0
draw x=350 y=-80 z=0
draw x=350 y=-125 z=0
draw x=-350 y=-125 z=0
draw x=-350 y=-80 z=0
draw x=-150 y=-80 z=0
draw x=-150 y=80 z=0
draw x=-350 y=80 z=0
draw x=-350 y=125 z=0
draw x=350 y=125 z=0
draw x=350 y=80 z=0
draw x=150 y=80 z=0
draw x=150 y=0 z=0
extrude-surface z=100
store b
clear
cylinder r=40 h=100
rotate x=90
translate x=250 y=-175 z=0
add b
store b
clear
cylinder r=40 h=100
rotate x=90
translate x=-250 y=-175 z=0
add b
store b
clear
r-joint-i
rotate x=90
translate x=250 y=0 z=0
add b
store b
clear
r-joint-i+1
rotate x=90
translate x=-250 y=0 z=0
add b
store-link 3r.l2
;       ****************************************
;
;       THIRD LINK
;       ****************************************
;
clear
box x=400 y=150 z=100
store b
```

```
clear
r-joint-i
rotate x=90
translate x=125 y=0 z=0
add b
store b
clear
r-joint-i+1
rotate x=90
translate x=-125 y=0 z=0
add b
store-link 3r.l3
end
```

## FILE: H (HOME POSITION VECTOR)

```
h 0.00 0.00 0.00 0.00 0.00 0.00
```

## FILE: DEMO.CMD

```
set-echo 1
minimal-step 6
translate-object box1 x=1875 z=750
grasp 3r box1
drive 3r 180 65 -65 0 0 0
release 3r box1
drive 3r 180 70 -80 0 0 0
drive 3r h
translate-object box2 x=1875 z=750
grasp 3r box2
drive 3r 180 45 -45 0 0 0
release 3r box2
drive 3r 180 50 -60 0 0 0
drive 3r h
translate-object box3 x=1875 z=750
grasp 3r box3
drive 3r 180 30 -30 0 0 0
release 3r box3
drive 3r 180 35 -45 0 0 0
drive 3r h
translate-object box4 x=1875 z=750
grasp 3r box4
drive 3r 180 17 -17 0 0 0
release 3r box4
drive 3r 180 22 -32 0 0 0
```

```
drive 3r h
translate-object box5 x=1875 z=750
grasp 3r box5
drive 3r 180 5 -5 0 0 0
release 3r box5
drive 3r 180 10 -20 0 0 0
drive 3r h
translate-object box6 x=1875 z=750
grasp 3r box6
drive 3r 180 -7 7 0 0 0
release 3r box6
drive 3r 180 -2 -8 0 0 0
drive 3r h
translate-object box7 x=1875 z=750
grasp 3r box7
drive 3r 180 -18.5 18.5 0 0 0
release 3r box7
drive 3r 180 -13.5 3.5 0 0 0
drive 3r h
translate-object box8 x=1875 z=750
grasp 3r box8
drive 3r 180 -31 31 0 0 0
release 3r box8
drive 3r 180 -41 41 0 0 0
drive 3r h
translate-object box9 x=1875 z=750
grasp 3r box9
drive 3r 180 -46 46 0 0 0
release 3r box9
drive 3r 180 -56 56 0 0 0
drive 3r h
translate-object box10 x=1875 z=750
grasp 3r box10
drive 3r 180 -67 67 0 0 0
release 3r box10
drive 3r 180 -77 77 0 0 0
drive 3r h
end
```

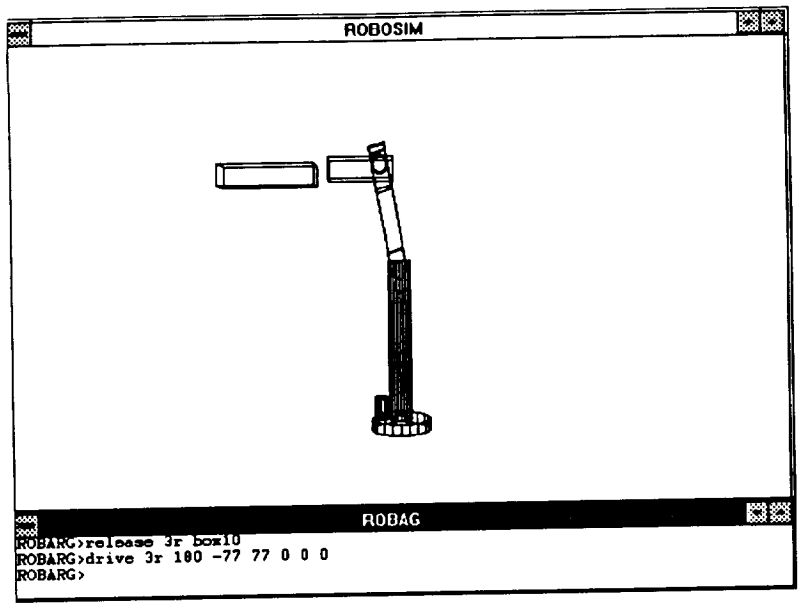


Figure E.5. Sample display (number 1) for Example 5.

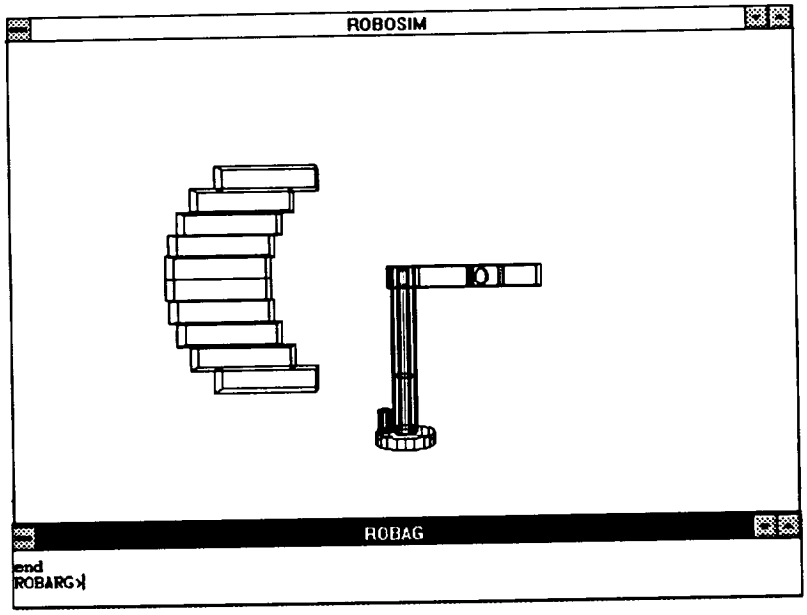


Figure E.6. Sample display (number 2) for Example 5.

## EXAMPLE 6: 6-AXIS OVERHEAD ROBOT

This example demonstrates a 6-axis overhead robot used for displaying information about ROBOSIM. To run the program, first copy the ROBO386.EXE file from your ROBOSIM root directory to this directory. You will also need to run the INIT.BAT file from your root directory to tell the simulation program where the FONTS and DRIVERS directories are located. After performing these steps, the program may be run by typing robo386 at the DOS prompt. Then the command file, DEMO.CMD, can be

loaded and run by typing 'load demo.cmd' at the robag> prompt. The overhead robot is named "hr". It may be moved with commands entered from the interactive window by use of the 'drive' command. The first joint is prismatic, the other five joints are revolute. The simulation program may be exited by typing 'exit' at the robag> prompt. You may wish to print out a listing of the HANG-ROB.DAT file to see how the robot is constructed and how the link joints are assigned.

**FILE: SETUP.CMD**

```
;       ****************************************
;                       SETUP.CMD
;       ****************************************
;                   COLOR DEFINITIONS
;       ****************************************
define-color red 1.00 0.00 00.0
define-color green 0.00 1.00 0.00
define-color blue 0.00 0.00 1.00
define-color black 0.00 0.00 0.00
define-color purple 1.00 0.00 1.00
define-color cyan 0.00 1.00 1.00
define-color yellow 1.00 1.00 0.00
define-color white 1.00 1.00 1.00
define-color rose 0.70 0.20 0.20
define-color gold 0.50 0.50 0.20
define-color lt-grey 0.75 0.75 0.75
;       ****************************************
;                   VIEW COMMANDS
;       ****************************************
look-at 0 0 0
look-from 0 10000 0
;       ****************************************
;       OVERHEAD (HANGING) ROBOT - HR
;       ****************************************
make-agent hr hang-rob cyan
translate-object hr y=-1000 z=2500
;       ****************************************
;       ROBOSIM
;       ****************************************
make-robosim-object robosim robosim.dat robosim yellow
translate-object robosim x=-100 z=10800
define-grasping-point robosim 0 0 0 0 0 0
;       ****************************************
;       FORMATION OF CHARACTER BLOCK: USED FOR MODELING OF
;       ****************************************
make-robosim-object uses1 uses1.dat uses1 white
```

```
translate-object uses1 x=-700 z=10300
;       ****************************************
;       FORMATION OF CHARACTER BLOCK: ROBOTS/POSITIONERS AND
;       ****************************************
make-robosim-object uses2 uses2.dat uses2 white
translate-object uses2 x=-500 z=-10200
;       ****************************************
;       FORMATION OF CHARACTER BLOCK: OBJECTS IN WORKSPACE
;       ****************************************
make-robosim-object uses3 uses3.dat uses3 white
translate-object uses3 x=-800 z=-10700
;       ****************************************
;       FORMATION OF CHARACTER BLOCK: AND FOR SIMULATION OF
;       ****************************************
make-robosim-object uses4 uses4.dat uses4 white
translate-object uses4 x=-700 z=-11200
;       ****************************************
;       FORMATION OF CHARACTER BLOCK: ACTION SEQUENCES
;       ****************************************
make-robosim-object uses5 uses5.dat uses5 white
translate-object uses5 x=-1300 z=-11700
;       ****************************************
;       FORMATION OF CHARACTER BLOCK: PERFORMED BY ROBOTS
;       ****************************************
make-robosim-object uses6 uses6.dat uses6 white
translate-object uses6 x=-900 z=-12200
;       ****************************************
;       COMBINE INTO ONE COMPOSITE OBJECT
;       ****************************************
set-collision-check 0
end
```

**FILE: HANG-ROB.DAT**

```
;       ****************************************
;       HANG-ROB.DAT
;       MODELING AN OVERHEAD ROBOT
;       ****************************************
;       WORKCELL COORDINATE FRAMES
;       ****************************************
set-nface n=10
clear
f-joint-i
store b
clear
```

```
f-joint-i+1
rotate z=90
add b
store-link hr.loc
;       ****************************************
;       BASE LINK
;       ****************************************
clear
box x=7800 y=400 z=200
store b
clear
f-joint-i
rotate z=90
add b
store b
clear
p-joint-i+1
rotate y=90
rotate x=90
translate x=0 y=0 z=-100
add b
store-link hr.l0
;       ****************************************
;       FIRST LINK
;       ****************************************
clear
box x=800 y=400 z=200
store b
clear
cylinder r=50 h=200
rotate y=90
translate x=500 y=0 z=0
add b
store b
clear
cylinder r=200 h=200
translate x=0 y=0 z=-200
add b
store b
clear
p-joint-i
rotate y=90
rotate x=90
translate x=0 y=0 z=100
add b
```

```
store b
clear
r-joint-i+1
rotate x=180
rotate z=90
translate x=0 y=0 z=-300
add b
store-link hr.ll
;       ****************************************
;
;       SECOND LINK
;       ****************************************
;
clear
cylinder r=200 h=200
store b
clear
box x=300 y=200 z=25
translate x=0 y=0 z=-112.5
add b
store b
clear
box x=25 y=200 z=200
translate x=137.5 y=0 z=-225
add b
store b
clear
box x=25 y=200 z=200
translate x=-137.5 y=0 z=-225
add b
store b
clear
cylinder r=50 h=200
rotate y=90
translate x=250 y=0 z=-275
add b
store b
clear
r-joint-i
rotate x=180
rotate z=90
translate x=0 y=0 z=100
add b
store b
clear
r-joint-i+1
rotate x=90
```

```
rotate z=90
translate x=0 y=0 z=-275
add b
store-link hr.l2
;       ****************************************
;
;       THIRD LINK
;       ****************************************
;
clear
box x=800 y=200 z=200
store b
clear
cylinder r=100 h=200
translate x=-500 y=0 z=0
add b
store b
clear
cylinder r=50 h=200
translate x=-500 y=0 z=200
add b
store b
clear
r-joint-i
translate x=350 y=0 z=0
add b
store b
clear
r-joint-i+1
translate x=-500 y=0 z=0
add b
store-link hr.l3
;       ****************************************
;
;       FOURTH LINK
;       ****************************************
;
clear
box x=1000 y=100 z=100
store b
clear
cylinder r=50 h=100
rotate y=90
translate x=-550 y=0 z=0
add b
store b
clear
r-joint-i
rotate z=90
```

```
translate x=600 y=0 z=0
add b
store b
clear
r-joint-i+1
rotate y=-90
rotate x=-90
translate x=-600 y=0 z=0
add b
store-link hr.l4
;       ****************************************
;
;       FIFTH LINK
;       ****************************************
;
clear
cylinder r=50 h=100
store b
clear
cylinder r=50 h=200
rotate y=90
;rotate z=90
translate x=0 y=0 z=-100
add b
store b
clear
r-joint-i
rotate x=180
rotate z=90
translate x=0 y=0 z=50
add b
store b
clear
r-joint-i+1
rotate y=90
rotate x=90
translate x=0 y=0 z=-100
add b
store-link hr.l5
;       ****************************************
;
;       SIXTH LINK
;       ****************************************
;
clear
cylinder r=50 h=50
store b
clear
cylinder r=10 h=400
```

```
translate x=0 y=0 z=-225
add b
store b
clear
r-joint-i
rotate x=90
translate x=0 y=0 z=75
add b
store b
clear
r-joint-i+1
rotate x=90
translate x=0 y=0 z=-100
add b
store-link hr.l6
end
```

**FILE: DEMO.CMD**

```
minimal-step 100
drive hr 3200 0 90 135 0 45
drive hr -2750 167 150 -45 0 0
drive hr 3225 167 150 -45 0 0
make-object box2 box 6000 10 3000 green
translate-object box2 z=-800
drive hr 1325 116 18 68 0 -45
drive hr 3200 116 18 68 0 -45
make-object box1 box 1900 10 400 purple
translate-object box1 z=950
drive hr 2000 90 -20 130 0 75
make-robosim-object nasavu nasavu.dat nasavu red
translate-object nasavu x=-1700 y=-3000 z=2300
drive hr 3200 116 18 68 0 -45
translate-object robosim z=-10000
minimal-step 2
drive hr 3225 167 150 -45 0 0
translate-object uses1 z=-10000
translate-object uses2 z=10000
translate-object uses3 z=10000
translate-object uses4 z=10000
translate-object uses5 z=10000
translate-object uses6 z=10000
drive hr 3200 90 -10 100 0 -90
minimal-step 100
drive hr 1975 117 -4 94 0 -90
```

```
make-agent t3-mm t3-mm cyan
translate-object t3-mm z=10000
minimal-step 4
drive t3-mm 0 0 0 0 0 1500
link-objects uses robosim uses1 uses2 uses3 uses4 uses5 uses6 box1 box2
define-grasping-point uses 0 0 0 0 0 0
grasp hr uses
drive hr 1975 -90 -4 94 0 0
release hr uses
minimal-step 100
drive hr 3200 0 90 90 0 0
make-robosim-object please please.dat please white
translate-object please x=-1500
make-robosim-object for-next for-next.dat for-next white
translate-object for-next x=-1600 z=-500
minimal-step 4
drive t3-mm 0 0 0 0 0 1000
end
```

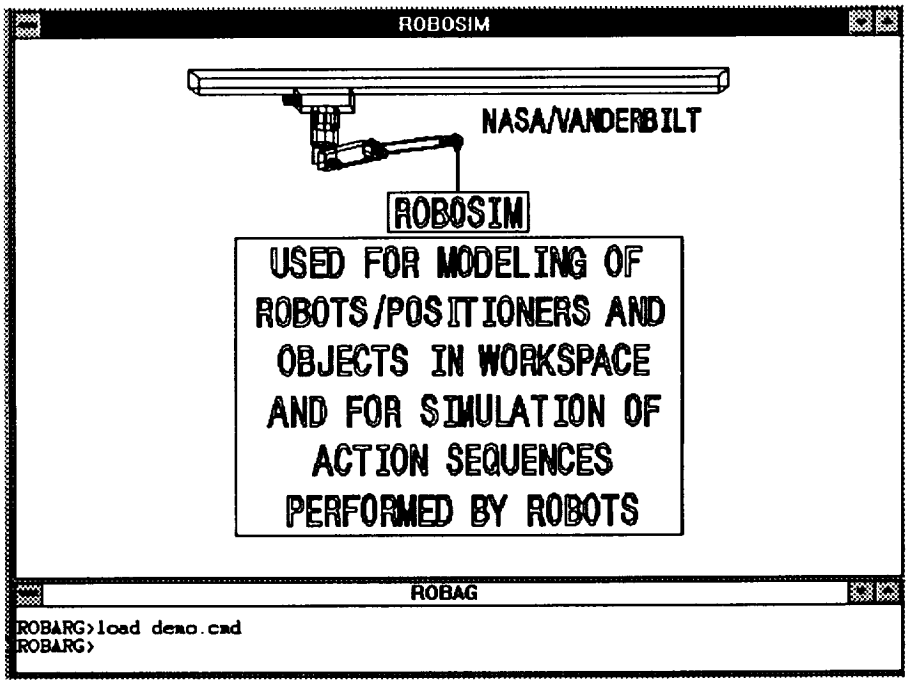


Figure E.7. Sample display (number 1) for Example 6.

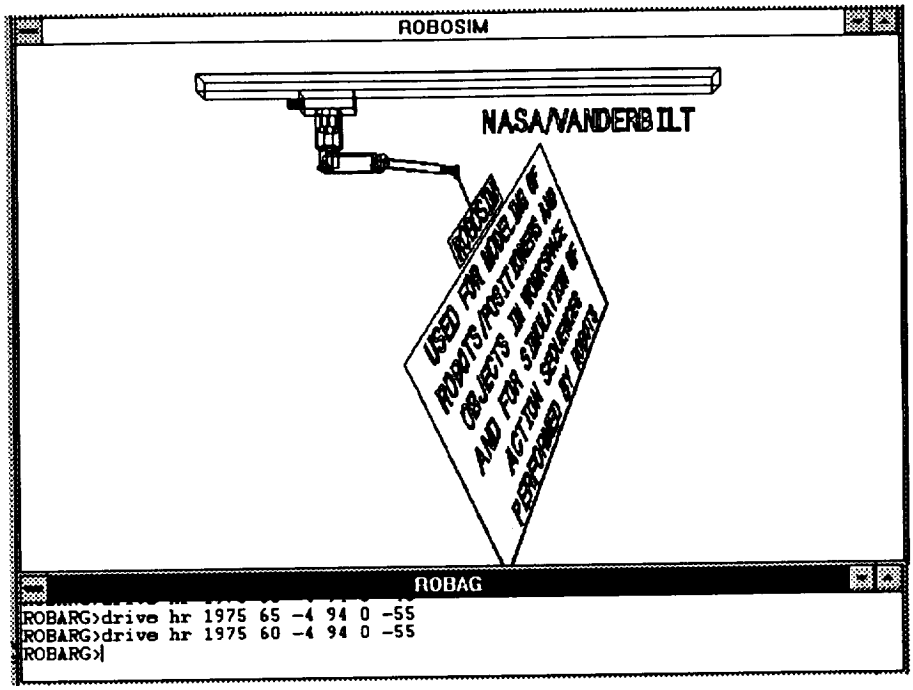


Figure E.8. Sample display (number 2) for Example 6.

## EXAMPLE 7: WATERBLAST REFURBISHMENT OF SRB COMPONENT

This example demonstrates the use of a Cincinnati Milacron T3 robot in the waterblast refurbishment of a solid rocket booster component at NASA's Marshall Space Flight Center. To run the program, first copy the ROBO386.EXE file from your ROBOSIM root directory. Also, you must run the INIT.BAT file from your root directory in order that the simulator program will know the path to the FONTS and DRIVERS directories. Following these two steps, type 'robo386' at the DOS prompt to enter the simulator. Then, at the robag> prompt, type 'load demo.cmd' to run the demonstration. When the demonstration is completed, the simulator may be exited by typing 'exit' at the robag> prompt. You may wish to print out a copy of the T3-MM.DAT file to see how the T3 robot is constructed and how the link joints are assigned.

## FILE: SETUP.CMD

```
;       ************************************************************
;
;                          SETUP.CMD
;       ************************************************************
;
;                       COLOR DEFINITIONS
;       ************************************************************
;
define-color red 1.00 0.00 00.0
define-color green 0.00 1.00 0.00
define-color blue 0.00 0.00 1.00
define-color black 0.00 0.00 0.00
```

```
define-color purple 1.00 0.00 1.00
define-color cyan 0.00 1.00 1.00
define-color yellow 1.00 1.00 0.00
define-color white 1.00 1.00 1.00
define-color rose 0.70 0.20 0.20
define-color gold 0.50 0.50 0.20
define-color lt-grey 0.6 0.6 0.6
;       ****************************************************************
;
;                       VIEW COMMANDS
;       ****************************************************************
;
look-at 0 0 1300
look-from 0 10000 1300
;       ****************************************************************
;
;               WATERBLAST REFURBISHMENT OF SRB - TITLE
;       ****************************************************************
;
make-robosim-object srb-titl srb-titl.dat srb-titl white
translate-object srb-titl x=2000 z=3450
;       ****************************************************************
;
;                       NASA/VANDERBILT LOGO
;       ****************************************************************
;
make-robosim-object nasavu nasavu3.dat nasavu red
translate-object nasavu x=-1650 z=-1000
;       ****************************************************************
;
;                       ROBOSIM LOGO
;       ****************************************************************
;
make-robosim-object robosim robosim3.dat robosim yellow
translate-object robosim x=-2300 z=-650
;       ****************************************************************
;
;                       HOME POSITION VECTOR
;       ****************************************************************
;
load-positions pos
;       ****************************************************************
;
;                       T3 ROBOT
;       ****************************************************************
;
make-agent t3 t3-mm cyan
translate-object t3 x=-6300 z=200
minimal-step 100
drive t3 home
translate-object t3 x=4000
;       ****************************************************************
;
;                       STAND FOR ROBOT
;       ****************************************************************
;
make-robosim-object stand stand.dat stand lt-grey
translate-object stand x=-2300 z=-200
;       ****************************************************************
;
```

```
;                    TURN TABLE AND SRB COMPONENT
;       ************************************************************
;
make-agent t-table t-table lt-grey
make-robosim-object srb srb.dat srb green
define-grasping-point srb 0 0 0 0 0 0
grasp t-table srb
translate-object t-table x=1500 z=-200
;       ************************************************************
;
;                    WATER JET
;       ************************************************************
;
make-robosim-object jet jet.dat jet blue
define-grasping-point jet 0 0 0 0 0 0
rotate-object jet y=-63.43
translate-object jet x=-4500 z=500
set-collision-check 0
end
```

**FILE: SRB.DAT**

```
;       ****************************************************
;
;               SRB.DAT
;       ****************************************************
;
;               MODELING OF SOLID ROCKET
;               BOOSTER AFT SKIRT
;       ****************************************************
;
move x=2000 y=0 z=0
draw x=1250 y=0 z=2500
translate x=0 y=0 z=600
rev-surface
store-file srb
end
```

**FILE: T-TABLE.DAT**

```
;       ****************************************************
;
;               T-TABLE.DAT
;       ****************************************************
;
;               MODELING OF ROTARY TURNTABLE
;       ****************************************************
;
;               WORKCELL COORDINATE FRAMES
;       ****************************************************
;
clear
f-joint-i
f-joint-i+1
store-link t-table.loc
```

```
;       ***************************************************
;
;               BASE LINK
;       ***************************************************
;
clear
cylinder r=2000 h=400
store b
clear
f-joint-i
translate x=0 y=0 z=-200
add b
store b
clear
r-joint-i+1
translate x=0 y=0 z=200
add b
store b
store-link t-table.l0
;       ***************************************************
;
;               FIRST LINK
;       ***************************************************
;
clear
cylinder r=2000 h=200
store b
clear
r-joint-i
translate x=0 y=0 z=-100
add b
store b
clear
r-joint-i+1
translate x=0 y=0 z=100
add b
store b
store-link t-table.l2
end
```

**FILE: JET.DAT**

```
cylinder r=12.7 h=500
translate x=0 y=0 z=250
store b
clear
move x=0 y=0 z=0
draw x=200 y=0 z=344
draw x=200 y=.1 z=344
```

```
add b
store b
clear
move x=0 y=0 z=0
draw x=200 y=0 z=344
draw x=200 y=.1 z=344
rotate z=60
add b
store b
clear
move x=0 y=0 z=0
draw x=200 y=0 z=344
draw x=200 y=.1 z=344
rotate z=120
add b
store b
clear
move x=0 y=0 z=0
draw x=200 y=0 z=344
draw x=200 y=.1 z=344
rotate z=180
add b
store b
clear
move x=0 y=0 z=0
draw x=200 y=0 z=344
draw x=200 y=.1 z=344
rotate z=240
add b
store b
clear
move x=0 y=0 z=0
draw x=200 y=0 z=344
draw x=200 y=.1 z=344
rotate z=300
add b
store-file jet
end
```

**FILE: STAND.DAT**

```
clear
cylinder r=100 h=300
translate x=0 y=550 z=150
store b
```

```
clear
cylinder r=100 h=300
translate x=-476 y=-275 z=150
add b
store b
clear
cylinder r=100 h=300
translate x=476 y=-275 z=150
add b
store b
clear
cylinder r=650 h=100
translate x=0 y=0 z=350
add b
store-file stand
end
```

**FILE: T3-MM.DAT**

```
;*********************************************************
;
;  MODELLING THE T3 ROBOT
;
;*********************************************************
SET-NFACE  N=15
CLEAR
F-JOINT-I
F-JOINT-I+1
STORE-LINK T3.LOC
;
;*********************************************************
;
;  MODELLING THE ROBOT'S BASE LINK
;
CLEAR
TRUNCATED-CONE RL=406.4 RU=254 H=914.4
TRANSLATE X=0 Y=0 Z=393.7
STORE B
CLEAR
CYLINDER R=482.6 H=88.9
TRANSLATE X=0 Y=0 Z=44.45
F-JOINT-I
ADD B
STORE B
CLEAR
```

```
R-JOINT-I+1
TRANSLATE X=0 Y=0 Z=1509.27
ADD B
STORE-LINK T3.L0
;
;*****************************************************
;
;
;  MODELLING THE ROBOT'S FIRST LINK
;
;
SET-COLOR I=3
CLEAR
CYLINDER R=279.4 H=101.6
TRANSLATE X=0 Y=0 Z=50.8
STORE B
CLEAR
BOX X=254 Y=50.8 Z=508
TRANSLATE X=0 Y=-101.6 Z=355.6
STORE C
TRANSLATE X=0 Y=203.2 Z=0
ADD C
ADD B
TRANSLATE X=0 Y=0 Z=-508
STORE B
CLEAR
CYLINDER R=203.2 H=355.6
TRANSLATE X=0 Y=0 Z=-304.8
R-JOINT-I+1
ROTATE X=90.
R-JOINT-I
ADD B
STORE-LINK T3.L1
;
;*****************************************************
;
;
;  MODELLING ROBOT LINK 2
;
SET-COLOR I=12
CLEAR
STORE B
STORE C
STORE D
BOX X=152.4 Y=152.4 Z=152.4
TRANSLATE X=-76.2 Y=0 Z=0
CYLINDER R=76.2 H=152.4
```

```
ROTATE Z=-30.
R-JOINT-I+1
TRANSLATE X=1016 Y=0 Z=0
STORE B
CLEAR
BOX X=1016 Y=152.4 Z=152.4
TRANSLATE X=355.6 Y=76.2 Z=0
ADD B
R-JOINT-I
STORE-LINK T3.L2
;
;**********************************************
;
;
;  MODELLING ROBOT LINK 3
;
SET-COLOR I=5
CLEAR
STORE B
STORE C
STORE D
BOX X=762 Y=152.4 Z=152.4
TRANSLATE X=-381 Y=0 Z=0
ROTATE Z=-11.54
TRANSLATE X=-127 Y=0 Z=0
BOX X=254 Y=152.4 Z=152.4
TRANSLATE X=-127 Y=0 Z=0
R-JOINT-I+1
TRANSLATE X=1016 Y=0 Z=0
CYLINDER R=76.2 H=152.4
R-JOINT-I
STORE-LINK T3.L3
;
;**********************************************
;
;
;  MODEL FOURTH ROBOT LINK
;
SET-COLOR I=13
CLEAR
STORE B
STORE C
STORE D
CYLINDER R=63.5 H=96.52
R-JOINT-I
STORE B
CLEAR
```

```
R-JOINT-I+1
CYLINDER R=63.5 H=101.6
ROTATE X=-90.
TRANSLATE X=203.2 Y=0 Z=0
STORE C
CLEAR
BOX X=76.2 Y=76.2 Z=76.2
TRANSLATE X=101.6 Y=0 Z=0
ADD B
ADD C
STORE-LINK T3.L4
;
;*****************************************
;
;
;  BUILD 5TH DEGREE OF FREEDOM WRIST PITCH
;
SET-COLOR I=7
CLEAR
STORE B
STORE C
STORE D
CYLINDER R=88.9 H=127
TRANSLATE X=0 Y=0 Z=76.2
BOX X=177.8 Y=177.8 Z=25.4
TRANSLATE X=0 Y=0 Z=165.1
STORE B
CLEAR
BOX X=25.4 Y=177.8 Z=215.9
TRANSLATE X=-63.5 Y=0 Z=44.45
STORE C
TRANSLATE X=127 Y=0 Z=0
ADD B
ADD C
ROTATE Y=90.
STORE B
CLEAR
CYLINDER R=63.5 H=76.2
TRANSLATE X=0 Y=0 Z=114.3
ADD B
STORE B
CLEAR
R-JOINT-I+1
ROTATE X=90.
R-JOINT-I
ROTATE Z=90.
```

```
ADD B
STORE-LINK T3.L5
;
;*************************************************
;
;
;  BUILD 6TH DEGREE OF FREEDOM
;
SET-COLOR I=11
CLEAR
STORE B
STORE C
STORE D
F-JOINT-I+1
TRANSLATE X=0 Y=0 Z=44.45
CYLINDER R=101.6 H=76.2
TRANSLATE X=0 Y=0 Z=63.5
CYLINDER R=76.2 H=50.8
TRANSLATE X=0 Y=0 Z=330.2
R-JOINT-I
STORE-LINK T3.L6
;
;*************************************************
;
;
;  BUILD END-EFFECTOR
;  LOOKS LIKE A CYLINDRICAL ROD
;
CLEAR
STORE B
STORE C
STORE D
;
;  BUILD A COORDINATE FRAME FOR END OF TOOL TIP
;
;LOAD-FILE AXIS.TCP
;
;  DEFINE ARM CONTROL POINT
;
F-JOINT-I+1
ROTATE Z=90.
TRANSLATE X=0 Y=0 Z=177.8
STORE B
CLEAR
CYLINDER R=12.7 H=152.4
TRANSLATE X=0 Y=0 Z=76.2
ADD B
```

```
STORE B
CLEAR
F-JOINT-I
ADD B
STORE-LINK T3.L7
END
```

**FILE: POS**

```
home 0.00 45.00 -135.00 0.00 90.00 0.00
h 0.00 0.00 0.00 0.00 0.00 0.00
pos9 0.00 77.50 -57.50 -45.00 90.00 0.00
pos8 0.00 83.50 -77.00 -35.00 90.00 0.00
pos7 0.00 88.00 -98.50 -14.50 90.00 0.00
pos6 0.00 86.50 -111.30 -1.00 90.00 0.00
pos5 0.00 81.50 -121.50 13.00 90.00 0.00
pos4 0.00 75.00 -128.50 22.00 90.00 0.00
pos3 0.00 55.00 -136.00 58.50 90.00 0.00
pos2 0.00 37.50 -134.00 70.75 90.00 0.00
pos1 0.00 25.00 -130.00 73.00 90.00 0.00
pos10 0.00 59.50 -15.50 -70.00 90.00 0.00
```

**FILE: DEMO.CMD**

```
set-echo 1
minimal-step 1
drive t3 90 45 -135 0 90 0
drive t3 pos1
translate-object jet x=4000
minimal-step 1
drive t-table 180 0 0 0 0 0
translate-object jet x=-4000
minimal-step 100
drive t3 pos2
translate-object jet x=4072.5 z=245
minimal-step 1
drive t-table 360 0 0 0 0 0
translate-object jet x=-4000
minimal-step 100
drive t3 pos3
translate-object jet x=4072.5 z=245
minimal-step 1
drive t-table 540 0 0 0 0 0
translate-object jet x=-4000
minimal-step 100
```

```
drive t3 pos4
translate-object jet x=4072.5 z=245
minimal-step 1
drive t-table 720 0 0 0 0 0
translate-object jet x=-4000
minimal-step 100
drive t3 pos5
translate-object jet x=4072.5 z=245
minimal-step 1
drive t-table 900 0 0 0 0 0
translate-object jet x=-4000
minimal-step 100
drive t3 pos6
translate-object jet x=4072.5 z=245
minimal-step 1
drive t-table 1080 0 0 0 0 0
translate-object jet x=-4000
minimal-step 100
drive t3 pos7
translate-object jet x=4072.5 z=245
minimal-step 1
drive t-table 1260 0 0 0 0 0
translate-object jet x=-4000
minimal-step 100
drive t3 pos8
translate-object jet x=4072.5 z=245
minimal-step 1
drive t-table 1440 0 0 0 0 0
translate-object jet x=-4000
minimal-step 100
drive t3 pos9
translate-object jet x=4072.5 z=245
minimal-step 1
drive t-table 1620 0 0 0 0 0
translate-object jet x=-4000
minimal-step 100
drive t3 pos10
translate-object jet x=4072.5 z=245
minimal-step 1
drive t-table 1800 0 0 0 0 0
translate-object jet x=-4000
drive t3 90 70 -40 -45 90 0
drive t3 home
end
```

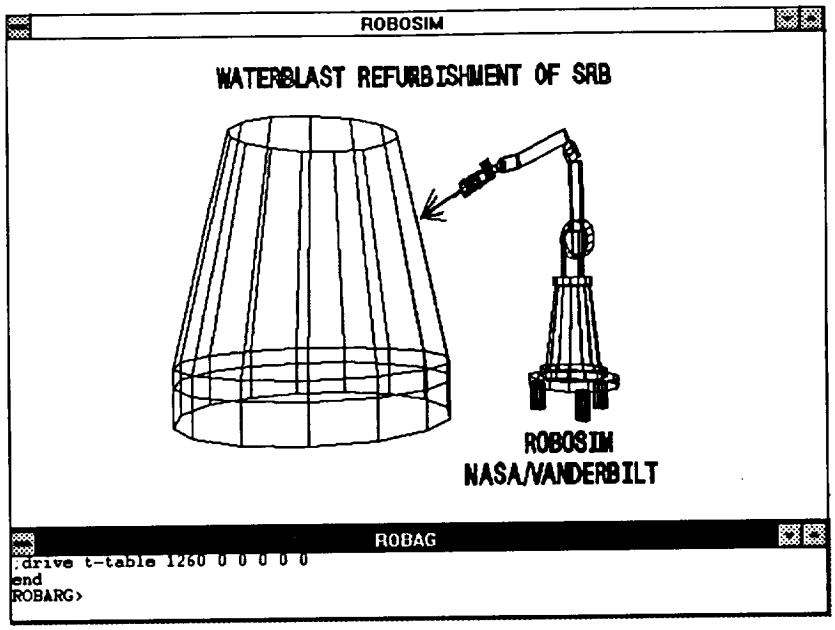


Figure E.9. Sample display for Example 7.

## EXAMPLE 8: SPACE STATION CONCEPT WITH SERVICING ROBOT

This example contains all of the files necessary to simulate the space station and run a demonstration of movement of the solar panels and servicing robot. The program may be run directly from the a-drive. To run the program, change to the ROBOSIM directory and type 'init' at the <A:\ROBOSIM> prompt. This will set the path to the FONTS directory and the DRIVERS directory. Next type 'robo386' at the <A:\ROBOSIM> prompt. After loading the necessary files, the space station simulation will appear on the screen. To run the demonstration, type 'load demo.cmd' at the ROBAG> prompt.

If you wish to load the files onto your hard disk and run the program from there, you will have to edit the INIT.BAT file accordingly. For example, if the hard disk directory used is <C:\ROBOSIM> then you merely need to change the drive designation from 'a' to 'c' in the two lines of the INIT.BAT file. If you use a directory other than ROBOSIM on your hard disk, then the directory designation must be changed also. For example, if the hard disk directory used is <C:\SSTATION>, then you must change the drive and directory designations from 'a:\robosim...' to c:\sstation...' in the two lines of the INIT.BAT file.

The robot and solar panels may be moved by issuing appropriate commands from the ROBAG> prompt. The servicing robot is called 'r'. The rotatable solar panels attached to the main trust assembly are called 'c1' and 'c2'. The rear and forward main solar panels are called 'mc1' and 'mc2'. The box on top of the main trust near the left-hand end is called b1. Refer to your ROBOSIM manual for a listing and description of the various commands that may be used to move the robot and/or solar panels.

If you already have ROBOSIM installed on your computer, you may wish to install the space station files in a subdirectory there. If so, you do not need to transfer the FONTS and DRIVERS directories to this subdirectory since they already exist in your ROBOSIM directory. Also, you can delete the ROBO386.EXE, INIT.BAT, and GO32.EXE files since they exist, as well, in your ROBOSIM directory.

**FILE: SETUP.CMD**

```
;    ************************************************************
;
;                SETUP.CMD
;    ************************************************************
;
;    COLOR DEFINITIONS
;    ****************************************
;
define-color red 1.00 0.00 00.0
define-color green 0.00 1.00 0.00
define-color blue 0.00 0.00 1.00
define-color black 0.00 0.00 0.00
define-color purple 1.00 0.00 1.00
define-color cyan 0.00 1.00 1.00
define-color yellow 1.00 1.00 0.00
define-color white 1.00 1.00 1.00
define-color rose 0.70 0.20 0.20
define-color gold 0.50 0.50 0.20
define-color lt-grey 0.75 0.75 0.75
;    ****************************************
;    VIEW COMMANDS
;    ****************************************
;
look-at 0 80 0
look-from 250 250 250
;       ****************************************
;       COLLISION CHECK
;       ****************************************
;
set-collision-check 0
;       ****************************************
;       ROBOT
;       ****************************************
;
make-agent r robot purple
translate-object r y=100
;       ****************************************
;       MAIN STRUCTURE AND LIVING MODULES
;       ****************************************
;
make-robosim-object main main.dat main yellow
make-robosim-object livinmod livinmod.dat livinmod red
;       ****************************************
;
```

```
;       ROTATABLE SOLAR PANELS ATTACHED TO MAIN TRUST
;       ****************************************
;
make-agent c1 cells green
make-agent c2 cells cyan
translate-object c2 y=-80
;       ****************************************
;       REAR AND FORWARD MAIN SOLAR PANELS
;       ****************************************
;
make-agent mc1 maincls1 blue
make-agent mc2 maincls2 white
;       ****************************************
;
;       COMMUNICATION DISHES
;       ****************************************
;
make-robosim-object dishes dishes.dat dishes white
;       ****************************************
;
;       BOX (MOVED BY ROBOT)
;       ****************************************
;
make-object b1 box 5 5 5 cyan
translate-object b1 x=5 y=5 z=2.5
end
```

**FILE: SUBCELLS.DAT**

```
;   This file creates the fixed set of cells attached
;   to the subframe
;
cylinder r=.05 h=20
rotate x=90
translate x=0 y=10 z=0
store b
clear
;
box x=5 y=20 z=.1
translate x=2.55 y=10 z=0
store d
rotate y=180
add d
add b
store d
;
clear
box x=10.2 y=.1 z=8
translate x=0 y=-.05 z=0
add d
translate x=0 y=.1 z=0
```

```
rotate x=-90
store-file subcells
end
```

**FILE: BELT.DAT**

```
;    This files constructs the conveyor belt attached
;    to the bottom of the main support trust
;    Some of the materials on the belt are also constructed
;
box x=6 y=160  z=.1
translate x=5  y=80  z=-9.95
store b
clear
;
box x=4  y=6  z=4
translate x=5  y=53  z=-7.95
add b
store b
clear
;
box x=5  y=4  z=6
translate x=5  y=63  z=-6.95
add b
store b
clear
;
box x=5  y=8  z=5
translate x=5  y=97  z=-7.45
add b
store b
clear
;
box x=4 y=3 z=3
translate x=5 y=105 z=-8.45
add b
end
```

**FILE: ROBOT.DAT**

```
;    This file constructs a robot arm
;
;
f-joint-i
translate x=0 y=100 z=0
```

```
f-joint-i+1
store-link robot.loc
;
;    Build robot base
;
clear
f-joint-i
store d
clear
box x=.5 Y=160 Z=.5
translate x=2.5 Y=80 z=0
store b
rotate y=180
add b
rotate y=-90
translate x=0 Y=0 Z=-5
store b
clear
p-joint-i+1
rotate x=90
rotate y=-90
translate x=-1 Y=80 Z=-5
add b
add d
store-link robot.l0
;
;    Build robot's first link
;
clear
p-joint-i
rotate x=90
box x=6 Y=8 Z=2
store b
clear
box x=1 y=1 Z=4
translate x=0 Y=0 z=3
add b
store b
clear
r-joint-i+1
translate x=0 y=0 z=5.5
add b
store-link robot.l1
;
;    Build robot's second link
```

```
;
clear
cylinder r=.5 h=1
rotate y=90
r-joint-i
store b
clear
box x=1 y=1 Z=4
rotate x=90
translate x=0 y=-2.5 z=0
store c
clear
r-joint-i+1
rotate x=90
translate x=0 y=-5 z=0
add c
add b
store-link robot.l2
;
;    Build third link
;
clear
cylinder r=.5 h=1
rotate y=90
r-joint-i
rotate y=90
store b
clear
box x=1 Y=1 Z=10
rotate x=90
translate x=0 y=5.5 z=0
store c
clear
r-joint-i+1
rotate x=-90
rotate z=90
translate x=0 y=11 z=0
add c
add b
store-link robot.l3
;
;    Build fourth link
;
clear
cylinder r=.5 h=1
```

```
r-joint-i
rotate x=90
store b
clear
box x=1 y=1 Z=8
translate x=0 y=0 z=4.5
store c
clear
r-joint-i+1
rotate x=90
translate x=0 y=0 z=9
add b
add c
store-link robot.l4
;
;    Build fifth link
;
clear
cylinder r=.5 h=1
r-joint-i
rotate x=90
store b
clear
box x=1 Y=1 Z=3
translate x=0 y=0 z=2
add b
store b
clear
r-joint-i+1
translate x=0 y=0 z=3.5
add b
store-link robot.l5
;
;    Build sixth link
;
clear
box x=1 y=2 z=.2
translate x=0 y=0 z=.1
store b
clear
box x=1 y=.2 z=.8
translate x=0 y=-.5 z=.6
store c
translate x=0 y=1 z=0
add c
```

```
add b
store b
clear
cylinder r=1 h=4
rotate x=90
translate x=0 y=0 z=1.5
add b
r-joint-i
store b
clear
f-joint-i+1
translate x=0 y=0 z=.01
add b
store-link robot.l6
;
;    Build end effector
;
clear
f-joint-i+1
translate x=0 y=0 z=.01
f-joint-i
store-link robot.l7
end
```

**FILE: MAINCLS1.DAT**

```
;    Subframe and attachment
;
clear
cylinder r=2 h=4
translate x=0 y=0 z=2
store d
clear
execute-file 2cubes.dat
store c
translate x=0 y=20 z=0
add c
translate x=0 y=20 z=0
add c
translate x=-5 y=0 z=5
store c
;
;    Bottom set of cells for the subframe
;
clear
```

```
execute-file subcells.dat
rotate z=-90
translate x=0 y=15 z=-5
store b
translate x=0 y=40 z=0
add b
add c
store c
;
;    Add a box in the subframe
;
clear
box x=8 y=8 z=8
translate x=0 y=5 z=-1
add c
store c
clear
move x=7.05 y=0 z=0
draw x=-7.05 y=0 z=0
draw x=-7.05 y=.001 z=0
store b
rotate z=90
add b
rotate z=45
rotate y=90
translate x=-5 y=15 z=0
store b
translate x=10 y=0 z=0
add b
store b
translate x=0 y=40 z=0
add b
add c
store c
;
;    Add forward maincells (fixed with respect to subframe)
;
clear
execute-file maincells.dat
translate x=0 y=15 z=0
add c
rotate x=90
translate x=0 y=0 z=4
store c
clear
```

```
cylinder r=2 h=4
translate x=0 y=0 z=2
add c
store c
clear
f-joint-i
f-joint-i+1
store-link maincell.loc
clear
f-joint-i
store b
clear
r-joint-i+1
rotate x=90
translate x=5 y=0 z=-5
add b
store-link maincell.l0
clear
r-joint-i
add c
store c
clear
r-joint-i+1
rotate x=-90
rotate z=90
translate x=0 y=0 z=59
add c
store-link maincell.l1
clear
;
;    Add rear maincells (rotable with respect to subframe)
;
execute-file maincells.dat
rotate y=90
r-joint-i
store c
clear
f-joint-i+1
translate x=0 y=0 z=.1
add c
store-link maincell.l2
end
```

**FILE: 2CUBES.DAT**

```
;ECHO OFF
;
; This file draws two cubes attached to each other
; with X=Y=Z=10. dimensions.
; The four diagonal ends of the cubes are at:
;       X=0.   Y=0.   Z=0.
;       X=10.  Y=0.   Z=-10.
;       X=0.   Y=20.  Z=0.
;       X=10.  Y=20.  Z=-10.
; The output is stored in file 2CUBES.OUT
;
MOVE X=0.,Y=0.,Z=0.
DRAW X=10.,Y=0.,Z=0.
draw x=10 y=.01 z=0
TRANSLATE X=-5.,Y=0.,Z=0.
STORE B
TRANSLATE X=5.,Y=0.,Z=0.
STORE C
TRANSLATE X=0.,Y=10.,Z=0.
ADD C
STORE C
CLEAR
LOAD B
ROTATE Z=90.
TRANSLATE X=0.,Y=5.,Z=0.
STORE D
TRANSLATE X=10.,Y=0.,Z=0.
ADD D
ADD C
STORE D
STORE C
CLEAR
MOVE X=0.,Y=0.,Z=0.
DRAW X=14.1,Y=0.,Z=0.
draw x=14.1 y=.01 z=0
ROTATE Z=-45.
TRANSLATE X=0.,Y=10.,Z=0.
ADD C
STORE C
ROTATE Y=90.
STORE B
TRANSLATE X=10.,Y=0.,Z=0.
ADD B
ADD C
STORE C
```

```
CLEAR
ADD D
TRANSLATE X=0.,Y=0.,Z=-10.
ADD C
STORE C
CLEAR
;
; Build reverse cube
;
ADD C
ROTATE Z=-180.
TRANSLATE X=10.,Y=20.,Z=0.
ADD C
;
;STORE-FILE 2CUBES.OUT
;store-file 2cubes
;
;LOOK-FROM X=20.,Y=10.,Z=30.
;LOOK-AT X=0.,Y=10.,Z=0.
;VIEW
;
END
```

**FILE: MAINCLS2.DAT**

```
;    Subframe and attachment
;
clear
cylinder r=2 h=4
translate x=0 y=0 z=2
store d
clear
execute-file 2cubes.dat
store c
translate x=0 y=20 z=0
add c
translate x=0 y=20 z=0
add c
translate x=-5 y=0 z=5
store c
;
;   Bottom set of cells for the subframe
;
clear
execute-file subcells.dat
```

```
rotate z=-90
translate x=0 y=15 z=-5
store b
translate x=0 y=40 z=0
add b
add c
store c
;
;    Add a box in the subframe
;
clear
box x=8 y=8 z=8
translate x=0 y=5 z=-1
add c
store c
clear
move x=7.05 y=0 z=0
draw x=-7.05 y=0 z=0
draw x=-7.05 y=.001 z=0
store b
rotate z=90
add b
rotate z=45
rotate y=90
translate x=-5 y=15 z=0
store b
translate x=10 y=0 z=0
add b
store b
translate x=0 y=40 z=0
add b
add c
store c
;
;    Add forward maincells (fixed with respect to subframe)
;
clear
execute-file maincells.dat
translate x=0 y=15 z=0
add c
rotate x=90
translate x=0 y=0 z=4
store c
clear
cylinder r=2 h=4
```

```
translate x=0 y=0 z=2
add c
store c
clear
f-joint-i
rotate x=180
rotate y=180
translate x=10 y=160 z=0
f-joint-i+1
store-link maincell.loc
clear
f-joint-i
store b
clear
r-joint-i+1
rotate x=90
translate x=5 y=0 z=-5
add b
store-link maincell.l0
clear
r-joint-i
add c
store c
clear
r-joint-i+1
rotate x=-90
rotate z=90
translate x=0 y=0 z=59
add c
store-link maincell.l1
clear
;
;    Add rear maincells (rotable with respect to subframe)
;
execute-file maincells.dat
rotate y=90
r-joint-i
store c
clear
f-joint-i+1
translate x=0 y=0 z=.1
add c
store-link maincell.l2
end
```

## FILE: MAIN.DAT

```
;    Construction of main trust assembly
;
execute-file 2cubes.dat
store b
translate x=0 y=20 z=0
add b
store c
translate x=0 y=40 z=0
add c
store d
translate x=0 y=80 z=0
add d
store d
;
;    Construction of conveyer belt in the main frame
;
clear
execute-file belt.dat
add d
store d
;
;    Add boxes and attach them to the main frame
;
clear
box x=4 y=3 z=6
translate x=0 y=5 z=-1
box x=4 y=3 z=8
translate x=12 y=43.5 z=-6
add d
store-file main
end
```

## FILE: CELLS.DAT

```
;    This file constructs the two single panel solar cells
;    attached to the main trust assemble
;
clear
f-joint-i
translate x=0 y=-100 z=0
f-joint-i+1
store-link cells.loc
clear
```

```
f-joint-i
store b
clear
r-joint-i+1
rotate x=-90
rotate z=90
translate x=2.5 y=15 z=-7.9
add b
store-link cells.l0
clear
;
set-nface n=10
cylinder r=2 h=5
cylinder r=1 h=5
translate x=0 y=0 z=2.5
store b
clear
;
cylinder r=.2 h=15
translate x=0 y=0 z=12.5
add b
store b
clear
;
box x=.6 y=12.2 z=.4
translate x=0 y=0 z=20.2
add b
store b
clear
;
cylinder r=.1 h=24
rotate x=90
store c
clear
;
box x=6 y=24 z=.2
translate x=3 y=0 z=0
store d
rotate y=180
add d
add c
rotate x=90
rotate z=90
translate x=0 y=0 z=32.4
add b
```

```
store b
;
rotate y=90
rotate y=90
store d
clear
r-joint-i
store c
clear
r-joint-i+1
add c
add d
store-link cells.l1
clear
end
```

**FILE: ATTBARS.DAT**

```
; This file draws the cylidrical structures that attaches
; the main frame with the two other sub-frames.
;
;LOOK-FROM X=18.,Y=-10.,Z=-5.
;LOOK-AT X=0.,Y=0.,Z=0.
;
BOX X=0.1,Y=5.9,Z=0.
TRANSLATE X=0.,Y=2.95,Z=0.
ROTATE X=32.5
TRANSLATE X=0.,Y=2.,Z=0.
STORE B
ROTATE Z=90.
ADD B
ROTATE Z=90.
ADD B
ROTATE Z=90.
ADD B
ROTATE Z=45.
;STORE-FILE ATTBARS.OUT
store-file attbars
END
```

**FILE: MAINCELL.DAT**

```
;    This file creates the main solar cells and the
;    attaching bars and support cylinders.
;
```

```
clear
cylinder r=5 h=120
rotate x=90
translate x=0 y=60 z=0
scale x=.25 y=.25 z=.25
store d
;
;    Solar cell plates
;
clear
box x=10 y=30 z=.2
translate x=6.3 y=15 z=0
store b
rotate y=180
add b
add d
translate x=0 y=.3 z=0
store d
;
;    Ends of each cell structure
;
clear
cylinder r=1.8 h=.3
box x=22.8 y=2.5 z=.3
rotate x=90
translate x=0 y=.15 z=0
store b
translate x=0 y=30.3 z=0
add b
add d
translate x=0 y=8 z=0
store d
;
;    Connecting cylinder between two cells
;
clear
cylinder r=1.6 h=8
rotate x=90
translate x=0 y=4 z=0
add d
store d
rotate x=180
add d
store d
clear
```

```
cylinder r=1.7 h=.1
store b
clear
rotate x=90
translate x=0 y=5 z=0
store b
translate x=0 y=-10 z=0
add b
add d
rotate z=90
end
```

**FILE: SUBFRAME.DAT**

```
;    Subframe and attachment
;
clear
cylinder r=2 h=4
translate x=0 y=0 z=2
store d
clear
execute-file 2cubes.dat
store c
translate x=0 y=20 z=0
add c
translate x=0 y=20 z=0
add c
translate x=-5 y=0 z=5
store c
;
;   Bottom set of cells for the subframe
;
clear
execute-file subcells.dat
rotate z=-90
translate x=0 y=15 z=-5
store b
translate x=0 y=40 z=0
add b
add c
store c
;
;    Add a box in the subframe
;
clear
```

```
box x=8 y=8 z=8
translate x=0 y=5 z=-1
add c
store c
clear
move x=7.05 y=0 z=0
draw x=-7.05 y=0 z=0
draw x=-7.05 y=.001 z=0
store b
rotate z=90
add b
rotate z=45
rotate y=90
translate x=-5 y=15 z=0
store b
translate x=10 y=0 z=0
add b
store b
translate x=0 y=40 z=0
add b
add c
store c
;
;    Add forward maincells (fixed with respect to subframe)
;
clear
execute-file maincells.dat
translate x=0 y=15 z=0
add c
rotate x=90
translate x=0 y=0 z=4
store c
clear
cylinder r=2 h=4
translate x=0 y=0 z=2
add c
store c
clear
f-joint-i
translate x=0 y=0 z=0
f-joint-i+1
store-link subframe.loc
clear
f-joint-i
store b
```

```
clear
r-joint-i+1
rotate x=90
translate x=5 y=0 z=-5
add b
store-link subframe.l0
clear
r-joint-i
add c
store c
clear
r-joint-i+1
rotate x=-90
rotate z=90
translate x=0 y=0 z=59
add c
store-link subframe.l1
clear
;
;    Add rear maincells (rotable with respect to subframe)
;
execute-file maincells.dat
rotate y=90
r-joint-i
store c
clear
f-joint-i+1
translate x=0 y=0 z=.1
add c
store-link subframe.l2
end
```

**FILE: LIVINMOD.DAT**

```
set-nface n=10
cylinder r=7 h=30
store b
clear
cylinder r=2 h=1
translate x=0 y=0 z=15.5
add b
store b
clear
cylinder r=2 h=1
translate x=0 y=0 z=-15.5
```

```
add b
;store c
translate x=0 y=8.5 z=0
store b
translate x=0 y=-17 z=0
add b
store b
;store-file livinmod
;end
clear
cylinder r=2 h=1
rotate x=90
translate x=0 y=0 z=-22
add b
store b
clear
cylinder r=2 h=1
rotate x=90
translate x=0 y=15 z=-22
add b
store b
clear
cylinder r=2 h=1
rotate x=90
translate x=0 y=-15 z=-22
add b
store b
clear
cylinder r=6 h=14
rotate x=90
translate x=0 y=7.5 z=-22
add b
store b
clear
cylinder r=6 h=14
rotate x=90
translate x=0 y=-7.5 z=-22
add b
store d
;store-file livinmod
;end
clear
cylinder r=5 h=10
translate x=0 y=0 z=5.5
cylinder r=2 h=1
```

```
translate x=0 y=0 z=6.5
sphere r=6
store b
clear
cylinder r=2 h=1
rotate x=90
translate x=0 y=6.5 z=0
store c
rotate z=90
add c
store c
rotate z=180
add c
add b
;store c
translate x=0 y=0 z=22
store c
clear
cylinder r=7 h=30
store b
clear
cylinder r=2 h=1
translate x=0 y=0 z=15.5
add b
store b
clear
cylinder r=2 h=1
translate x=0 y=0 z=-15.5
add b
add c
rotate x=90
translate x=0 y=0 z=23
add d
store d
;
clear
cylinder r=5 h=10
translate x=0 y=0 z=5.5
cylinder r=2 h=1
translate x=0 y=0 z=6.5
sphere r=6
store b
clear
cylinder r=2 h=1
rotate x=90
```

```
translate x=0 y=6.5 z=0
store c
rotate z=90
add c
store c
rotate z=180
add c
add b
store c
clear
cylinder r=2 h=1
translate x=0 y=0 z=-6.5
add c
rotate y=90
translate x=14 y=0 z=0
store c
rotate z=180
add c
translate x=0 y=0 z=-8.5
rotate x=90
translate x=0 y=0 z=23
add d
store b
clear
;
;
cylinder r=7 h=30
translate x=0 y=0 z=15.5
cylinder r=2 h=1
store c
clear
cylinder r=2 h=1
translate x=0 y=0 z=31.5
add c
translate x=0 y=8.5 z=0
store c
clear
cylinder r=7 h=22
translate x=0 y=0 z=11.5
cylinder r=2 h=1
store d
clear
cylinder r=2 h=1
translate x=0 y=0 z=23.5
add d
```

```
translate x=0 y=-8.5 z=0
add c
store c
clear
cylinder r=6 h=6
translate x=0 y=0 z=3.25
cylinder r=2 h=.5
translate x=0 y=0 z=7.25
rotate y=90
store d
rotate z=180
add d
translate x=0 y=-8.5 z=14
add c
translate x=0 y=0 z=30.5
add b
rotate y=-90
rotate z=180
translate x=0 y=70 z=-17
store-file livinmod
end
```

**FILE: DISHES.DAT**

```
;    This file constructs the two communication dishes attached
;    to the main support trust
clear
move x=0 y=0 z=0
draw x=0 y=7.2 z=0
draw x=0 y=7.2 z=.05
rotate x=24.6
translate x=0 y=-7.05 z=0
store b
rotate z=90
add b
store b
rotate z=180
add b
rotate z=45
store b
clear
cylinder r=.5 h=20
translate x=0 y=0 z=13
add b
store b
```

```
clear
move x=0 y=2.1 z=.5
draw x=0 y=1 z=5.12
draw x=.05 y=1 z=5.12
store c
rotate z=120
add c
store c
clear
move x=0 y=2.1 z=.5
draw x=0 y=1 z=5.12
draw x=.05 y=1 z=5.12
rotate z=240
add c
store c
clear
cylinder r=1 h=.1
translate x=0 y=0 z=5.25
add c
translate x=0 y=0 z=23
add b
store b
clear
move x=0 y=0.5 z=0
draw x=0 y=5.3 z=1.2
draw x=0 y=3.7 z=.8
draw x=0 y=2.1 z=.4
draw x=0 y=.5 z=0
rev-surface
translate x=0 y=0 z=23
add b
store b
translate x=5 y=25 z=0
store c
clear
add b
translate x=5 y=125 z=0
add c
store-file dishes
end
```

**FILE: DEMO.CMD**

```
look-at 0 80 0
look-from 250 250 250
```

```
minimal-step 1
drive mc1 -45 0 0 0 0 0
drive mc1 -45 90 0 0 0 0
drive mc2 45 0 0 0 0 0
drive mc2 45 -90 0 0 0 0
drive c1 90 0 0 0 0 0
drive c2 -90 0 0 0 0 0
look-at 0 0 0
look-from 200 0 0
drive mc1 0 0 0 0 0 0
minimal-step 3
drive mc1 90 0 0 0 0 0
drive mc1 90 90 0 0 0 0
minimal-step 1
look-at 0 160 0
look-from 200 160 0
drive mc2 0 0 0 0 0 0
minimal-step 3
drive mc2 90 0 0 0 0 0
drive mc2 90 90 0 0 0 0
look-at 0 80 0
look-from 120 80 120
drive c1 0 0 0 0 0 0 , drive c2 0 0 0 0 0 0
minimal-step 1
look-from -60 80 200
drive r 65.5 0 0 0 0 0
look-at 0 0 0
look-from -40 0 40
minimal-step 1
drive r 65.5 0 -80 10 40 0
grasp r b1
drive r 65.5 0 0 0 0 0
look-at 0 80 0
look-from -120 80 120
drive r -75 0 0 0 0 0
look-at 0 160 0
look-from -40 160 40
minimal-step 1
drive r -75 0 -80 10 40 0
release r b1
drive r -75 0 0 0 0 0
look-at 0 80 0
look-from -120 80 120
drive r 0 0 0 0 0 0
look-at 0 80 0
```

look-from 250 250 250
minimal-step 10
drive mc1 0 0 0 0 0 0 , drive mc2 0 0 0 0 0 0
end

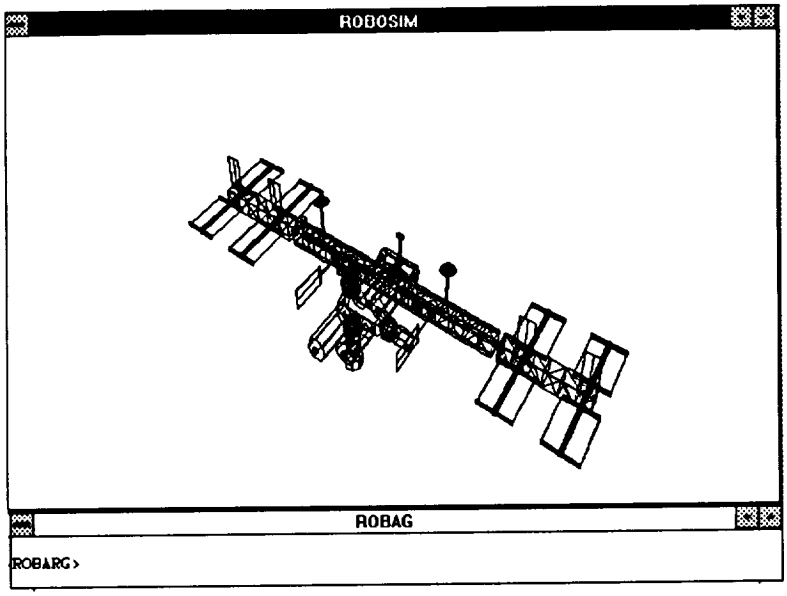


Figure E.10. Space station concept (sample display number 1), Example 8.

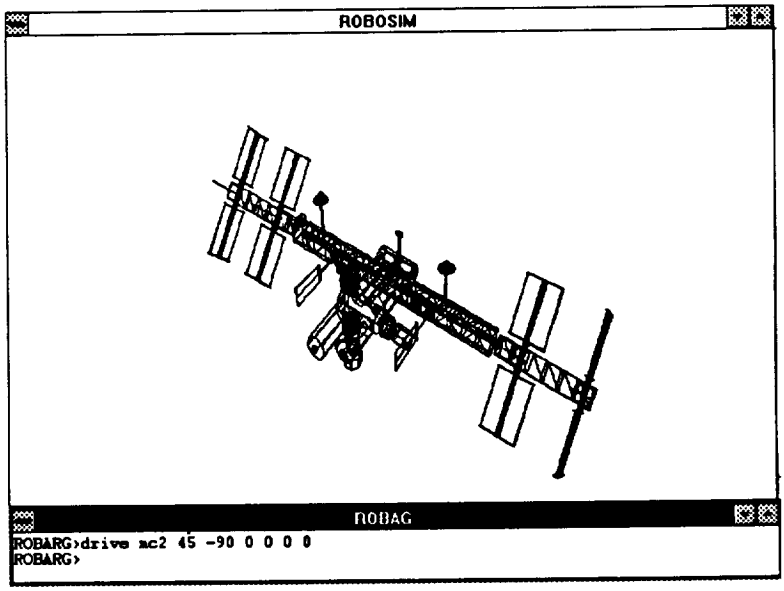


Figure E.11. Space station concept (sample display number 2), Example 8.

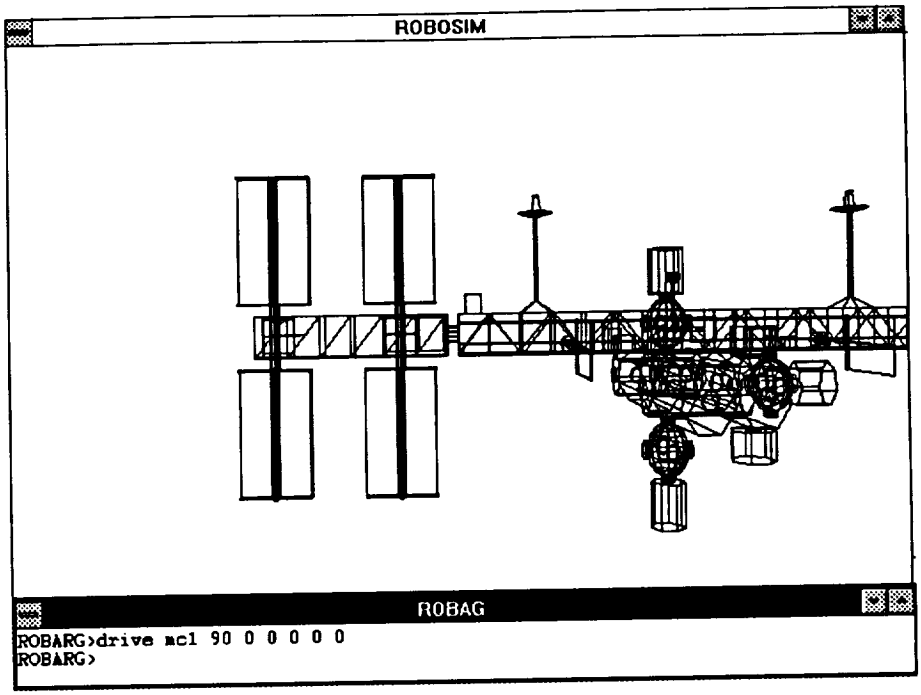


Figure E.12. Space station concept (solar panels), Example 8.

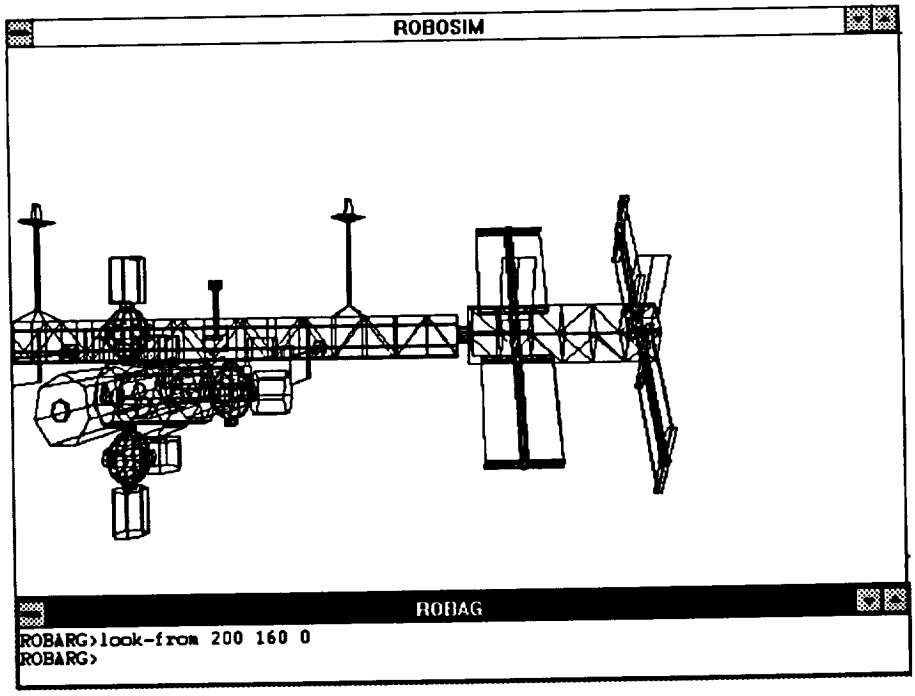


Figure E.13. Space station concept (solar panels - 2), Example 8.

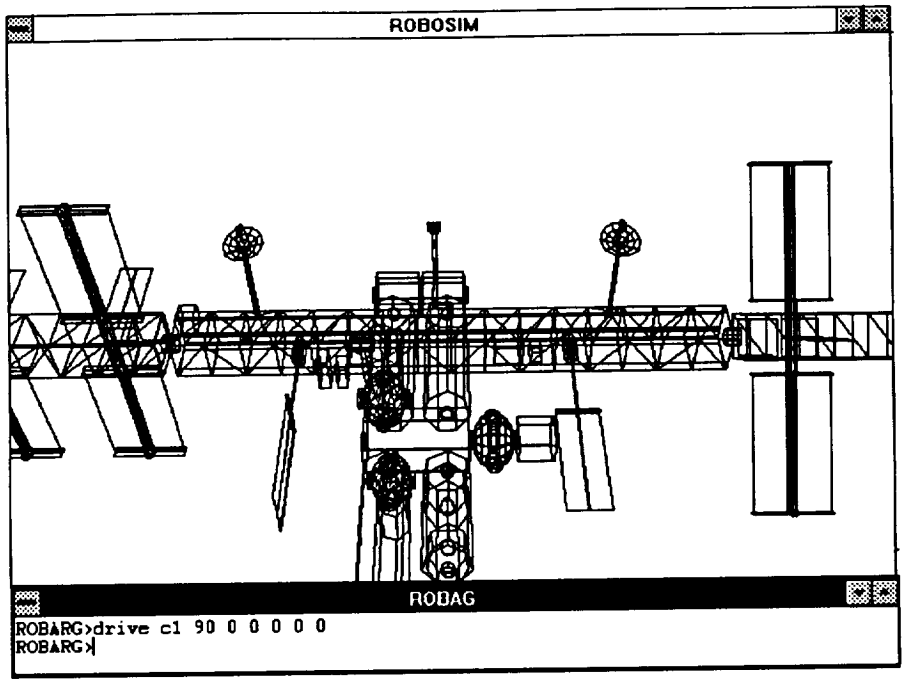


Figure E.14. Space station concept (sample display number 5), Example 8.

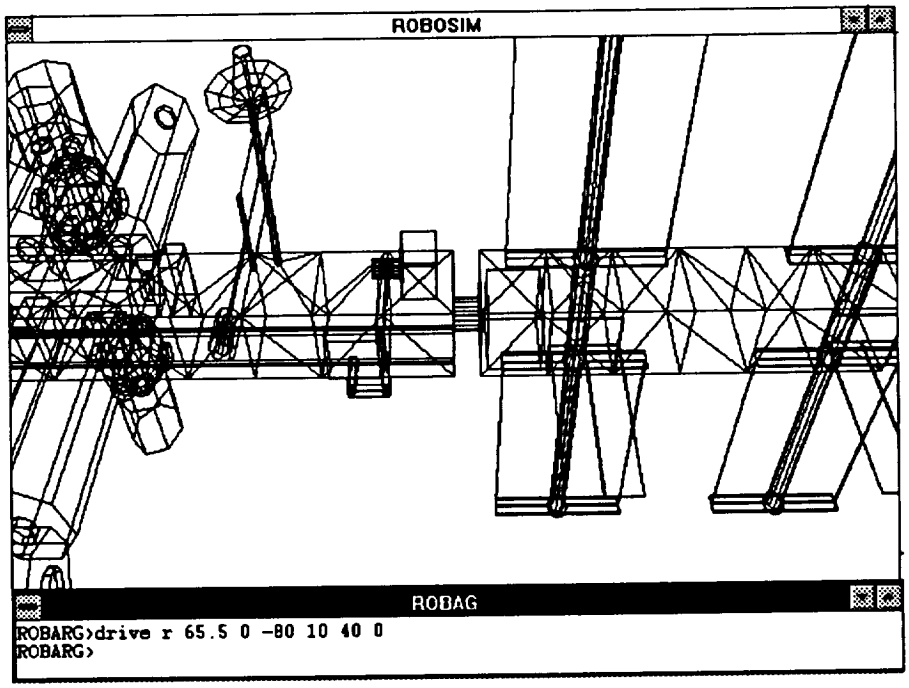


Figure E.15. Space station servicing robot (sample display number 1), Example 8.

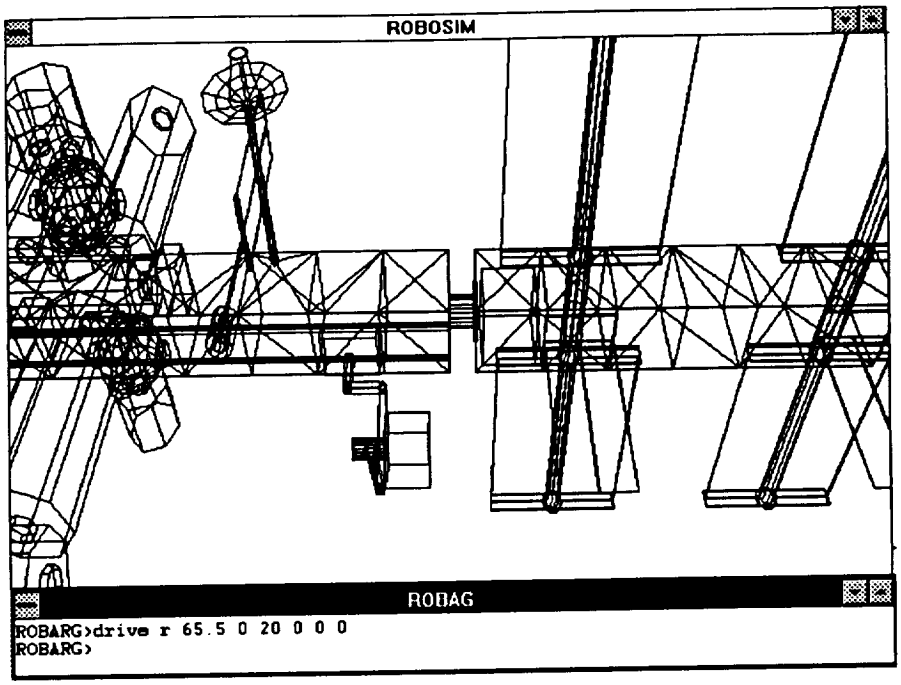


Figure E.16. Space station servicing robot (sample display number 2), Example 8.

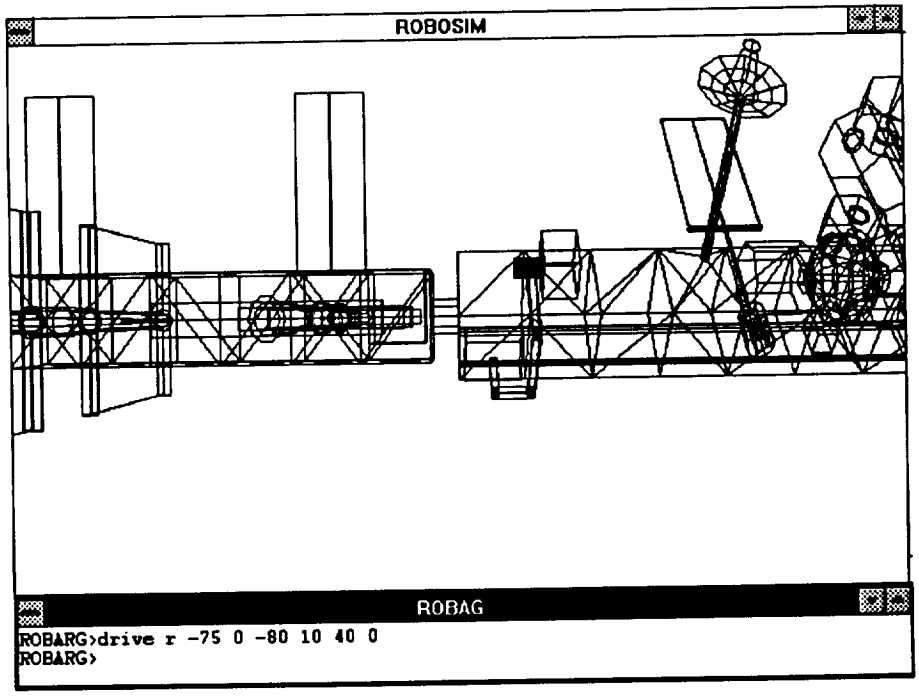


Figure E.17. Space station servicing robot (sample display number 3), Example 8.

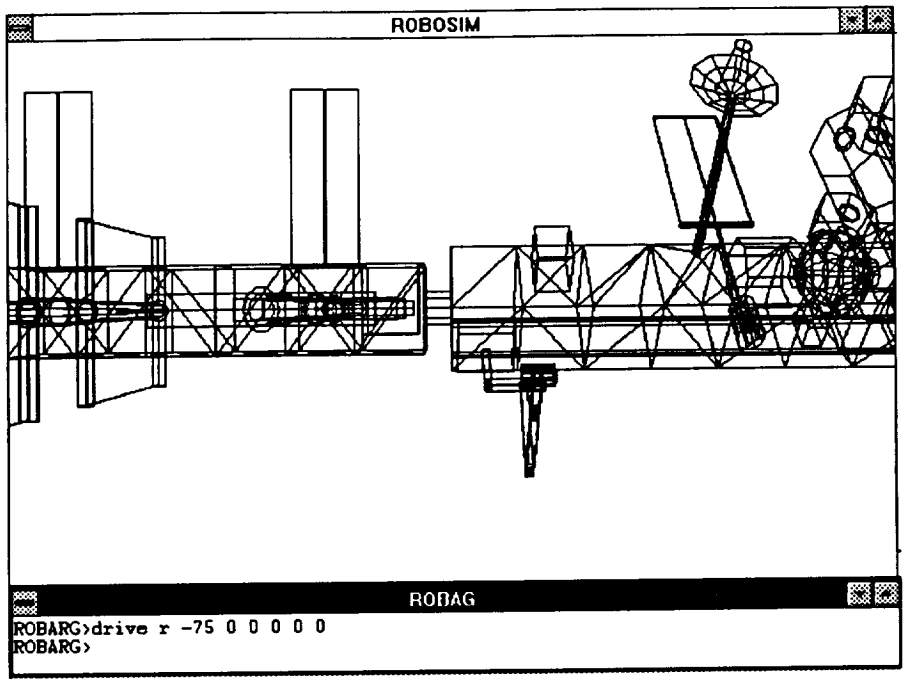


Figure E.18. Space station servicing robot (sample display number 4), Example 8.